```
Network Working Group                                         I. Bryskin
Internet-Draft                                       Huawei Technologies
Intended status: Informational                                    X. Liu
Expires: September 1, 2018                                         Jabil
                                                                A. Clemm
                                                                  Huawei
                                                             H. Birkholz
                                                          Fraunhofer SIT
                                                                 T. Zhou
                                                                  Huawei
                                                       February 28, 2018
```

Generalized Network Control Automation YANG Model
draft-bryskin-netconf-automation-yang-01

Abstract

This document describes a YANG data model for the Generalized Network Control Automation (GNCA), aimed to define an abstract and uniform semantics for NETCONF/YANG scripts in the form of Event-Condition-Action (ECA) containers.

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 September 1, 2018.

Copyright Notice

Table of Contents

1. Introduction

NETCONF/YANG has proved to be very successful in facilitating interactive network control paradigm, in which a network control application (controller) requests the network server to perform certain (re-)configurations, then, retrieves network states of interest, then asks for more (re-)configurations and so forth. This said, there are multiple use cases that require stringing network configuration requests together with conditioning their order of execution based on certain events detected in the network, as well as current, historical or predicted network states, and pushing such request batches/scripts to the network server in order to control the network close loop automation processes. The Generalized Network Control Automation (GNCA) YANG model introduced by this document defines an abstract and uniform semantics of such NETCONF/YANG scripts in the form of Event-Condition-Action (ECA) containers.

2. Purpose

The purpose of the Generalized Network Control Automation (GNCA) YANG model is to enable an environment allowing for manipulation of close loop network automation via configuration of abstract Event-Condition-Action (ECA) scripts. The model defines the semantics of said scripts; however, how the scripts are actually implemented by the server is not governed by the model. The server may, for example, based on pushed ECA configuration, generate and execute server specific scripts. How this is done is outside of the scope of this document.

Although not all event response behaviors could be delegated to the network server, there are many circumstances where it is highly desirable. For example:

a. Reaction to many network events is well understood and does not require additional information (such as broader or higher level network view or analytics input);

b. It is often imperative for the network to start acting as quickly as the event is detected. In other words, there might simply be no time for the network-controller communication;

c. A paradigm in which a controller micro-manages every network behavior does not scale. Simply put, there are always things that the network has to do autonomously (i.e. when the controller is not looking), which does not mean that the controller should have no say as to how said things need to be done;

d. Numerous important use cases and applications can benefit from ability to define in an abstract and uniformly way a programmable logic in one place (e.g. on the controller) and execute it someplace else (e.g. on the server).

The main objective of the GNCA is to generalize the ECA network control style, so that it could be applied to arbitrary network events/conditions and manipulate network auto-control of large variety of network resources. Such a generalization would allow, for example, conveying to the network a coherent/ordered/prioritized behavior for recovering from a network failure or failures affecting simultaneously multiple services, instruct the network how to fight rapidly developing congestions, what to do when power outage is detected and so forth. In fact, it could be argued that without some sort of close loop network control automation hierarchical network control realistically could not be achieved.

The GNCA YANG model could be also thought of as an attempt to generalize the smart filter subscription machinery by stating that sending a notification is one of possibly multiple actions that network could perform reacting to the specified smart trigger. According to "Smart filters for Push Updates - Problem Statement" [I-D.clemm-netconf-push-smart-filters-ps]:

"They [smart filters] 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."

In a nutshell GNCA facilitates an environment in which the network automation logic could be defined in a form of ECA scripts by a client, pushed to the network before the events of interest happen, and executed by the network after the events are detected.

Finally, according to the smart filters problem statement, the following smart filters will be considered:

"o Filters that involve freely programmable logic"

ECA scripts could serve as a vehicle to associate with a smart filter a complex freely programmable logic to detect the event of interest in the first place.

3. GNCA concepts and constructs

3.1. Policy Variable (PV)

Policy Variable (PV) is a structure to keep interim results/meta data during the execution of an ECA script. For example, a PV could be used as an output parameter of an RPC invoked by ECA1 to be used in a condition expression for ECA2.

With respect to ECAs the scope of a PV is either global or (ECA) local. Global PVs are permanent. They are kept in the top level PV container and are shared between all ECAs of the script. Local PVs are kept within the internal PV container of an ECA. Local PVs could be either dynamic - appear/disappear with start/stop of the ECA execution, or static - exist as long as the ECA is configured.

Each PV has the following attributes:

- Globally or ECA scope unique name;

- type - either pre-defined (e.g. Boolea, integer, uint64,etc.) or specified via XPath pointing to a data store node or a sub-tree of the required structure. For example, PV named "Link" could be associated with the "/TE_Topologies/TE_Links/TE_Link" XPath in accordance with the TE Topology model [I-D.ietf-teas-yang-te-topo] and hence be capable of accommodating the entire TE Link container;

- value - data stored in the PV structured according to the PV type.

The following operations are allowed with/on a PV:

- initialize (with a constant/enum/identity);

- assign (with contents of another same type PV);

- read (retrieve data store contents pointed by the specified same type XPath/sub-tree);

- write (modify same type CONFIG=TRUE data store state with the PV’s content/value);

- insert (PV’s content into a same type list);

- iterate (copy into PV one by one same type list elements) (See examples of definition of and operations with PVs in Section 5).

- function calls in a form of F(dst, src), where F is an identity of a function from extendable function library, dst and src are destination and source PVs respectively, the function’s input parameters, with the result returned in dst.

Arbitrary expressions with PVs are for future study.

PVs could be used as input/output of an ECA invoked RPC. PVs could also be a source of information sent to the client in notification messages.

PVs could be used in condition expressions (see Section 5).

The model structure for the Policy Variable is shown below:

```
   +--rw policy-variables
   |  +--rw policy-variable* [name]
   |     +--rw name     string
   |     +--rw (type-choice)?
   |     |  +--:(common)
   |     |  |  +--rw type?    identityref
   |     |  +--:(xpath)
   |     |     +--rw xpath?   string
   |     +--rw value?   <anydata>
```

3.2. ECA Event

ECA Event is any subscribable event notification either explicitly defined in a YANG module supported by the server or a smart filter conveyed to the server via smart filter subscription. Additionally, an event could be associated with a one-time or periodic timer.

Event notification contents become in the ECA context implicit local dynamic PVs with automatically generated names. Let’s assume Network_Failure_Is_Detected event is defined that carries to a subscriber the detected failure type (failureType), ID (failureID) and the affected by the failure TE link state (teLInk). In the context of the associated with the Networ_Failure_Is_Detected event ECA failureType, failureID and teLink are implicit local dynamic PVs that could be used in the embodied Condition and Action containers along with explicitly defined global and ECA local (static and/or dynamic) PVs.

One way to think of NETWONF/YANG Event Notification is as Remote Procedure Call-back, i.e. server->client directed RPC When a subscribable NETWONF/YANG Event and associated with it ECA is pushed to the server within an ECA script, the server is expected to interpret this as follows: take the contents of the event notification and execute the logic defined by the associated Condition-Action chain (as I (client) would do on receipt of the notification) in order to decide how to react to the event. Recall that the whole purpose of ECA scripts is to reduce as much as possible the client-server communication.

All events (specified in at least one ECA pushed to the server) are required to be constantly monitored by the server. One way to think of this is that the server subscribes to its own publications with respect to all events that are associated with at least one ECA.

3.3. ECA Condition

ECA Condition is evaluated to TRUE or FALSE logical expression. There are two ways how an ECA Condition could be specified:

- in a form of XPath expression;

- as a hierarchy of comparisons and logical combinations of thereof.

The former option allows for specifying a condition of arbitrary complexity as a single string with an XPath expression, in which pertinent PVs and data store states are referred to by their respective positions in the YANG tree.

The latter option allows for configuring logical hierarchies. The bottom of said hierarchies are primitive comparisons (micro-conditions) specified in a form of:

<arg1><relation><arg2>

> where arg1 and arg2 represent either constant/enum/identity, PV or pointed by XPath data store node or sub-tree,
>
> relation is one of the comparison operations from the set: ==, !=, >, <, >=, <=

Primitive comparisons could be combined hierarchically into macro-conditions via && and || logical operations.

Regardless of the choice of their specification, ECA Conditions are associated with ECA Events and evaluated only within event threads triggered by the event detection.

When an ECA Condition is evaluated to TRUE, the associated with it ECA Action is executed.

The model structure for the ECA Condition is shown below:

```
   +--rw conditions
   |  +--rw condition* [name]
   |     +--rw name                      string
   |     +--rw (expression-choice)?
   |        +--:(logical-operation)
   |        |  +--rw logical-operation-type?   identityref
   |        |  +--rw comparison-operation* [name]
   |        |  |  +--rw name                string
   |        |  |  +--rw comparision-type?   identityref
   |        |  |  +--rw arg1
   |        |  |  |  +--rw policy-argument
   |        |  |  |     +--rw type?                    identityref
   |        |  |  |     +--rw (argument-choice)?
   |        |  |  |        +--:(policy-constant)
   |        |  |  |        |  +--rw constant?                string
   |        |  |  |        +--:(policy-variable)
   |        |  |  |        |  +--rw policy-variable?         leafref
   |        |  |  |        +--:(local-policy-variable)
   |        |  |  |        |  +--rw local-policy-variable?   leafref
   |        |  |  |        +--:(xpath)
   |        |  |  |           +--rw xpath?                   string
   |        |  |  +--rw arg2
   |        |  |     +--rw policy-argument
   |        |  |        +--rw type?                    identityref
   |        |  |        +--rw (argument-choice)?
   |        |  |           +--:(policy-constant)
   |        |  |           |  +--rw constant?                string
   |        |  |           +--:(policy-variable)
   |        |  |           |  +--rw policy-variable?         leafref
   |        |  |           +--:(local-policy-variable)
   |        |  |           |  +--rw local-policy-variable?   leafref
   |        |  |           +--:(xpath)
   |        |  |              +--rw xpath?                   string
   |        |  +--rw sub-condition* [name]
   |        |     +--rw name    -> /gnca/conditions/condition/name
   |        +--:(xpath)
   |           +--rw condition-xpath?          string
```

The policy arguments arg1 and arg2 have the following structure:

```
   +--rw policy-argument
      +--rw type?                      identityref
      +--rw (argument-choice)?
         +--:(policy-constant)
         |  +--rw constant?                  string
         +--:(policy-variable)
         |  +--rw policy-variable?           leafref
         +--:(local-policy-variable)
         |  +--rw local-policy-variable?     leafref
         +--:(xpath)
            +--rw xpath?                     string
```

3.4. ECA Action

ECA Action is one of the following operations to be carried out by a server:

- (-re)configuration - modifying a CONFIG=TRUE data store state

- (re-)configuration scheduling - scheduling one time or periodic (re-)configuration in the future

- sending one time notification;

- adding/removing event notify subscription (essentially, the same action as performed when a client explicitly adds/removes a subscription)

- executing an RPC defined by a YANG module supported by the server (the same action as performed when a client interactively calls the RPC);

- performing operations and function calls on PVs (such as assign, read, insert, iterate, etc);

- starting/stopping timers;

- stopping current ECA;

- invoking another ECA;

- NO-ACTION action - meaningful only within ECA’s Cleanup Condition-Action list to indicate that the ECA’s Normal Condition-Action thread must be terminated as soon as one of the required Actions is rejected by the server (see more Section 3.4).

Two points are worth noting:

1. When a NETWONF/YANG RPC appears in an ECA Action body, the server is expected to interpret this as follows: execute the same logic, as when the client explicitly calls said RPC via NETCONF. For example, when TE_Tunnel_Path_Computation RPC is found in the currently executed Action, the server is expected to call its TE path computation engine and pass to it the specified parameters in the Action input.

2. When a "Send notification" action is configured as an ECA Action, the notification message to be sent to the client may contain not only elements of the data store (as, for example, YANG PUSH or smart filter notifications do), but also the contents of global and local PVs, which store results of arbitrary operations performed on the data store contents (possibly over arbitrary period of time) to determine, for example, history/evolution of data store changes, median values, ranges and rates of the changes, results of configured function calls and expressions, etc. - in short, any data the client may find interesting about the associated event with all the logic to compute said data delegated to the server.

Multiple ECA Condition/Action pairs could be combined into a macro-action.

Multiple ECA (macro-)Actions could be triggered by a single ECA event.

Any given ECA Condition or Action may appear in more than one ECAs.

The model structure for the ECA Action is shown below:

```
   +--rw actions
   |  +--rw action* [name]
   |     +--rw name              string
   |     +--rw action-element* [name]
   |     |  +--rw name                string
   |     |  +--rw action-type?        identityref
   |     |  +--rw (action-operation)?
   |     |     +--:(action)
   |     |     |  +--rw action-name?
   |     |     |          -> /gnca/actions/action/name
   |     |     +--:(content-moving)
   |     |     |  +--rw content-moving
   |     |     |     +--rw content-moving-type?   identityref
   |     |     |     +--rw src
   |     |     |     |  +--rw policy-argument
   |     |     |     +--rw dst
   |     |     |        +--rw policy-argument
   |     |     +--:(function-call)
   |     |     |  +--rw function-call
   |     |     |     +--rw function-type?   identityref
   |     |     |     +--rw src
   |     |     |     |  +--rw policy-argument
   |     |     |     +--rw dst
   |     |     |        +--rw policy-argument
   |     |     +--:(rpc-operation)
   |     |     |  +--rw rpc-operation
   |     |     |     +--rw name?              string
   |     |     |     +--rw nc-action-xpath?   string
   |     |     |     +--rw policy-variable* [name]
   |     |     |        +--rw name               string
   |     |     |        +--rw policy-argument
   |     |     +--:(notify-operation)
   |     |        +--rw notify-operation
   |     |           +--rw name?              string
   |     |           +--rw policy-variable* [name]
   |     |              +--rw name               string
   |     |              +--rw policy-argument
   |     +--rw time-schedule
   |        +--rw start?             yang:date-and-time
   |        +--rw repeat-interval?   string
```

3.5. ECA

An ECA container includes:

- Event name

- List of local PVs. As mentioned, local PVs could be configured as dynamic (their instances appear/disappear with start/stop of the ECA execution) or static (their instances exisr as long as the ECA is configured). The ECA input (the contents of the associated notification message, such as YANG PUSH or smart filter notification message) are the ECA's implict local dynamic PVs with automatically generated names

- Normal CONDITION-ACTION list: configured conditions each with associated actions to be executed if the condition is evaluated to TRUE

- Cleanup CONDITION-ACTION list: configured conditions/actions to be evaluated/executed in case any Action from the Normal CONDITION-ACTION list was attempted and rejected by the server. In other words, this list specifies the ECA cleanup/unroll logic after rejection of an Action from the Normal CONDITION-ACTION list. An empty Cleanup CONDITION-ACTION list signifies that the ECA's normal Actions should be executed regardless of whether the previously attempted ECA Actions were rejected or not by the server. Cleanup CONDITION-ACTION list containing a single NO-ACTION Action signifies that the ECA thread is to be immediately terminated on rejection of any attempted Action (without executing any cleanup logic)

4. Where ECA scripts are executed?

It could be said that the main idea of the GNCA is decoupling the place where the network control logic is defined from the place where it is executed. In previous sections of this document it is assumed that the network control logic is defined by a client and pushed to and executed by the network (server). This is accomplished via ECA scripts, which are essentially bunches of regular NETCONF style operations (such as get, set, call rpc) and event notifications glued together via Policy Variables, PVs. It is worth noting that said ECA scripts could be easily moved around and executed by any entity supporting the GNCA YANG model (i.e. capable of interpreting the ECA scripts). One interesting implication of this is that the ECA scripts could be executed by neither client nor server, but a 3d party entity, for instance, with a special focus on the control of a particular network domain or/and special availability of/proximity to information/ resources that could contribute to the network control decision process. For example, the ECA scripts could be pushed to a Path Computation Element (PCE) adopted to support the GNCA YANG model. Specialized ECA scripts could be fanned out to multiple specialized controllers to take care of different aspects of a network domain control.

Another interesting idea is to combine the GNCA with hierarchical T-SDN architecture. In particular, the ECA scripts conveyed by a client to a network orchestrator could be pushed (modified or unmodified) hierarchically down to lower level controllers. After all, the goal of the hierarchical T-SDN is to create a paradigm in which the higher level of a controller in the hierarchy, the broader (topologically and/or functionally) its control on the network and the lesser its involvement in the network’s micro-management in real time. On the other hand, it is desirable for a higher level controller to have a say as to how the subordinate controllers and, by extension, the network under control should deal with events and situations that are handled autonomously (i.e. without bothering the higher level controller in real time). The ECA scripts pushed down the T-SDN hierarchy may help to achieve this objective.

5. ECA script example

Consider a situation on a TE network when a network failure simultaneously affecting multiple TE tunnels. Normally, the TE network relies in this case on TE tunnels pre-configured protection/ restoration capabilities and lets the TE tunnels to auto-recover themselves independently from each other. However, this default behavior may not be desired in some configurations/use cases because:

a. Recovery procedures of a "greedy" TE tunnel may block the recovery of other TE tunnels;

b. Shared mesh protection/restoration schemes are in place

In such cases the network has to perform the recovery of failure affected TE tunnels as a coordinated process. Furthermore, it is quite common that network resources available for the dynamic recovery procedures are limited, in which case it is desirable to convey to the network the policy/order in which the TE tunnels should be recovered. Different policies may be considered, to name a few:

1. Recover as many TE tunnels as possible;

2. Recover TE tunnels in accordance with their importance/priority;

3. Recover all unprotected TE tunnels before recovering broken connections/LSPs of protected TE tunnels (because the latter can tolerate the failure hopefully until it is repaired).

Let’s describe an ECA script that could be pushed by the controller application instructing the network to perform multiple TE tunnel failure recovery according to policy (3) above.

Assumptions: it is assumed that in one or several YANG modules supported by the server the following is defined:

- Subscribable "Network_Failure_Is_Detected" event carrying in the notification message the detected failure type (failureType), ID (failureID) and the affected by the failure TE link ID (linkID);

- RPC "PathDependsOnLink" taking as an input a TE_Path and TE_Link_ID and returning in its output Boolean indicating whether or not the specified path goes through the link with the specified ID;

- RPC "ReplaceTunnelsAwayFromLink" taking as an input a list of TE tunnel key leafrefs and ID of to be avoided link, performing the tunnel replacement away from the link and returning no output.

Explicit (global) PVs:

- name: Yes type: Boolean

- name: lsp xpath: /TE_Tunnels/lsps/lsp

- name tunnel xpath: /TE_Tunnels/tunnels/te_tunnel

- name: unprotected_tunnels xpath: /TE_Tunnels/tunnels/te_tunnel/dependent_tunnels

- name protected_tunnels xpath: /TE_Tunnels/tunnels/te_tunnel/dependent_tunnels

Actions:

name: PopulateTunnelLists

body:

```
   lsp iterate xpath:/TE_Tunnels/lsps
   {
     rpc: PathDependsOnLink(<lsp>/rro, Yes);
     if(Yes == TRUE )
     {
       tunnel = <lsp>/parrent_tunnel;
       if(<tunnel>/protectionType == UNPROTECTED)
         <tunne>/tunnelName insert unprotected_tunnels
       if(<tunnel>/protectionType != UNPROTECTED)
         <tunne>/tunnelName insert protected_tunnels
     }
   }
```

name: RepairTunnels

Body:

```
   rpc: ReplaceTunnelsAwayFromLink(unprotected_tunnels, linkID);
   rpc: ReplaceTunnelsAwayFromLink(protected_tunnels, linkID);
```

ECA:

eventName: Network_Failure_Is_Detected;

eventParams: failureType, failureID, linkID

Condition: TRUE (always, every time)

Actions:

unprotected_tunnels = 0; protected_tunnels =0;

namedAction:PopulateTunnelLists;

namedAction:RepairTunnels

Note: RPC "PathDependsOnLink" is used in the example for simplicity. The RPC could be easily replaced by a scripted named action similar to PopulateTunnelListes .

6. Complete Model Tree Structure

The complete tree structure of the YANG model defined in this document is as follows:

```
   module: ietf-gnca
       +--rw gnca
          +--rw policy-variables
          |  +--rw policy-variable* [name]
          |     +--rw name     string
          |     +--rw (type-choice)?
          |     |  +--:(common)
          |     |  |  +--rw type?    identityref
          |     |  +--:(xpath)
          |     |     +--rw xpath?   string
          |     +--rw value?   <anydata>
          +--rw conditions
          |  +--rw condition* [name]
          |     +--rw name                      string
          |     +--rw (expression-choice)?
          |        +--:(logical-operation)
          |        |  +--rw logical-operation-type?   identityref
          |        |  +--rw comparison-operation* [name]
          |        |  |  +--rw name                string
          |        |  |  +--rw comparision-type?   identityref
          |        |  |  +--rw arg1
          |        |  |  |  +--rw policy-argument
          |        |  |  |     +--rw type?
    identityref
          |        |  |  |     +--rw (argument-choice)?
          |        |  |  |        +--:(policy-constant)
          |        |  |  |        |  +--rw constant?
    string
          |        |  |  |        +--:(policy-variable)
          |        |  |  |        |  +--rw policy-variable?
    leafref
          |        |  |  |        +--:(local-policy-variable)
          |        |  |  |        |  +--rw local-policy-variable?
    leafref
          |        |  |  |        +--:(xpath)
          |        |  |  |           +--rw xpath?
    string
          |        |  |  +--rw arg2
          |        |  |     +--rw policy-argument
          |        |  |        +--rw type?
    identityref
          |        |  |        +--rw (argument-choice)?
          |        |  |           +--:(policy-constant)
          |        |  |           |  +--rw constant?
    string
          |        |  |           +--:(policy-variable)
          |        |  |           |  +--rw policy-variable?
    leafref
```

```
          |        |  |           +--:(local-policy-variable)
          |        |  |           |  +--rw local-policy-variable?
    leafref
          |        |  |           +--:(xpath)
          |        |  |              +--rw xpath?
    string
          |        |  +--rw sub-condition* [name]
          |        |     +--rw name    -> /gnca/conditions/condition
   /name
          |        +--:(xpath)
          |           +--rw condition-xpath?          string
          +--rw actions
          |  +--rw action* [name]
          |     +--rw name              string
          |     +--rw action-element* [name]
          |     |  +--rw name                string
          |     |  +--rw action-type?        identityref
          |     |  +--rw (action-operation)?
          |     |     +--:(action)
          |     |     |  +--rw action-name?
          |     |     |          -> /gnca/actions/action/name
          |     |     +--:(content-moving)
          |     |     |  +--rw content-moving
          |     |     |     +--rw content-moving-type?   identityref
          |     |     |     +--rw src
          |     |     |     |  +--rw policy-argument
          |     |     |     |     +--rw type?
          |     |     |     |     |       identityref
          |     |     |     |     +--rw (argument-choice)?
          |     |     |     |        +--:(policy-constant)
          |     |     |     |        |  +--rw constant?
          |     |     |     |        |          string
          |     |     |     |        +--:(policy-variable)
          |     |     |     |        |  +--rw policy-variable?
    leafref
          |     |     |     |        +--:(local-policy-variable)
          |     |     |     |        |  +--rw local-policy-variable?
    leafref
          |     |     |     |        +--:(xpath)
          |     |     |     |           +--rw xpath?
          |     |     |     |                   string
          |     |     |     +--rw dst
          |     |     |        +--rw policy-argument
          |     |     |           +--rw type?
          |     |     |           |       identityref
          |     |     |           +--rw (argument-choice)?
          |     |     |              +--:(policy-constant)
          |     |     |              |  +--rw constant?
          |     |     |              |          string
          |     |     |              +--:(policy-variable)
          |     |     |              |  +--rw policy-variable?
    leafref
          |     |     |              +--:(local-policy-variable)
          |     |     |              |  +--rw local-policy-variable?
    leafref
          |     |     |              +--:(xpath)
          |     |     |                 +--rw xpath?
          |     |     |                         string
          |     |     +--:(function-call)
          |     |     |  +--rw function-call
          |     |     |     +--rw function-type?   identityref
          |     |     |     +--rw src
          |     |     |     |  +--rw policy-argument
          |     |     |     |     +--rw type?
          |     |     |     |     |       identityref
          |     |     |     |     +--rw (argument-choice)?
          |     |     |     |        +--:(policy-constant)
          |     |     |     |        |  +--rw constant?
          |     |     |     |        |          string
          |     |     |     |        +--:(policy-variable)
          |     |     |     |        |  +--rw policy-variable?
    leafref
          |     |     |     |        +--:(local-policy-variable)
          |     |     |     |        |  +--rw local-policy-variable?
    leafref
          |     |     |     |        +--:(xpath)
          |     |     |     |           +--rw xpath?
          |     |     |     |                   string
          |     |     |     +--rw dst
          |     |     |        +--rw policy-argument
          |     |     |           +--rw type?
          |     |     |           |       identityref
          |     |     |           +--rw (argument-choice)?
          |     |     |              +--:(policy-constant)
          |     |     |              |  +--rw constant?
          |     |     |              |          string
          |     |     |              +--:(policy-variable)
          |     |     |              |  +--rw policy-variable?
    leafref
          |     |     |              +--:(local-policy-variable)
          |     |     |              |  +--rw local-policy-variable?
    leafref
          |     |     |              +--:(xpath)
          |     |     |                 +--rw xpath?
          |     |     |                         string
          |     |     +--:(rpc-operation)
          |     |     |  +--rw rpc-operation
          |     |     |     +--rw name?              string
          |     |     |     +--rw nc-action-xpath?   string
          |     |     |     +--rw policy-variable* [name]
          |     |     |        +--rw name               string
          |     |     |        +--rw policy-argument
          |     |     |           +--rw type?
          |     |     |           |       identityref
          |     |     |           +--rw (argument-choice)?
          |     |     |              +--:(policy-constant)
          |     |     |              |  +--rw constant?
          |     |     |              |          string
          |     |     |              +--:(policy-variable)
          |     |     |              |  +--rw policy-variable?
    leafref
          |     |     |              +--:(local-policy-variable)
          |     |     |              |  +--rw local-policy-variable?
    leafref
          |     |     |              +--:(xpath)
          |     |     |                 +--rw xpath?
          |     |     |                         string
          |     |     +--:(notify-operation)
          |     |        +--rw notify-operation
          |     |           +--rw name?              string
          |     |           +--rw policy-variable* [name]
          |     |              +--rw name               string
          |     |              +--rw policy-argument
          |     |                 +--rw type?
          |     |                 |       identityref
          |     |                 +--rw (argument-choice)?
          |     |                    +--:(policy-constant)
          |     |                    |  +--rw constant?
          |     |                    |          string
          |     |                    +--:(policy-variable)
          |     |                    |  +--rw policy-variable?
    leafref
          |     |                    +--:(local-policy-variable)
          |     |                    |  +--rw local-policy-variable?
    leafref
          |     |                    +--:(xpath)
          |     |                       +--rw xpath?
          |     |                               string
          |     +--rw time-schedule
          |        +--rw start?             yang:date-and-time
          |        +--rw repeat-interval?   string
          +--rw ecas
          |  +--rw eca* [name]
          |     +--rw name                        string
          |     +--rw event-name                  string
          |     +--rw policy-variable* [name]
          |     |  +--rw name         string
          |     |  +--rw (type-choice)?
          |     |  |  +--:(common)
          |     |  |  |  +--rw type?        identityref
          |     |  |  +--:(xpath)
          |     |  |     +--rw xpath?       string
          |     |  +--rw value?       <anydata>
          |     |  +--rw is-static?   boolean
          |     +--rw condition-action* [name]
          |     |  +--rw name         string
          |     |  +--rw condition?   -> /gnca/conditions/condition/name
          |     |  +--rw action?      -> /gnca/actions/action/name
          |     +--rw cleanup-condition-action* [name]
          |     |  +--rw name         string
          |     |  +--rw condition?   -> /gnca/conditions/condition/name
          |     |  +--rw action?      -> /gnca/actions/action/name
          |     +---x start
          |     +---x stop
          |     +---x pause
          |     +---x resume
          |     +---x next-action
          |     +--ro execution* [id]
          |        +--ro id                               uint32
          |        +--ro oper-status?                     enumeration
          |        +--ro start-time?
          |        |       yang:date-and-time
          |        +--ro stop-time?
          |        |       yang:date-and-time
          |        +--ro next-scheduled-time?
          |        |       yang:date-and-time
          |        +--ro last-condition-action?
          |        |       -> ../../condition-action/name
          |        +--ro last-condition?
          |        |       -> ../../condition-action/condition
          |        +--ro last-action?
          |        |       -> ../../condition-action/action
          |        +--ro last-cleanup-condition-action?
          |                -> ../../cleanup-condition-action/name
          +--rw eca-scripts
          |  +--rw eca-script* [script-name]
          |     +--rw script-name    string
          |     +--rw eca* [eca-name]
          |        +--rw eca-name    -> /gnca/ecas/eca/name
          +--rw running-script?
                  -> /gnca/eca-scripts/eca-script/script-name
```

7. YANG Module

```
   <CODE BEGINS> file "ietf-gnca@2018-02-28.yang"
   module ietf-gnca {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-gnca";

     prefix "gnca";

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

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

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

        Editor:   Igor Bryskin
                  <mailto:Igor.Bryskin@huawei.com>

        Editor:   Xufeng Liu
                  <mailto:Xufeng_Liu@jabil.com>

        Editor:   Alexander Clemm
                  <mailto:ludwig@clemm.org>";

     description
       "Event Condition Action (ECA) model.";

     revision 2018-02-28 {
       description "Initial revision";
       reference "RFC XXXX";
     }

     /*
      * Typedefs
      */
     identity argument-type {
       description
         "Possible values are:
          constant, variable, or datastore state.";
     }

     identity comparison-type {
       description
         "Possible values are:
          equal, not-equal, greater, greater-equal, less, less-equal.";
     }

     identity logical-operation-type {
       description
         "Possible values are:
          not, or, and.";
     }

     identity function-type {
       description
         "Possible values are:
          plus, minus, mult, divide, remain.";
     }

     identity content-moving-operation-type {
       description
         "Possible values are:
          copy, iterate, insert.";
     }

     identity action-type {
       description
         "Possible values are:
          action, content-move, function-call, rpc, notify.";
     }

     identity policy-variable-type {
       description
         "Possible values are:
          boolean, int32, int64, uint32, uint64, string, etc.";
     }

     /*
      * Groupings
      */
     grouping policy-variable-attributes {
       description
         "Defining the policy variable attributes, including name, type
          and value. These attributes are used as part of the Policy
          Variable (PV) definition.";
       leaf name {
         type string;
         description
           "A string to uniquely identify a Policy Variable (PV), either
            globally for a global PV, or within the soope of ECA for a
```

```
             local PV.";
        }
        choice type-choice {
          description
            "The type of a policy variable may be either a common
             primative type like boolean or a type from existing
             schema node referenced by an XPath string.";
          case common {
            leaf type {
              type identityref {
                base policy-variable-type;
              }
              description
                "A common policy variable type, defined as an
                 identity.";
            }
          }
          case xpath {
            leaf xpath {
              type string;
              description
                "A XPath string, referencing a schema node, whose
                 type is used as the type of the policy variable.";
            }
          }
        }
        anydata value {
          description
            "The value of the policy variable, in a format that is
             determined by the policy type.";
        }
      } // policy-variable-attributes

      grouping policy-argument {
        description
          "Defining a policy argument, which can be used in a comparison
           or an action.";
        container policy-argument {
          description
            "Containing the attributes of a policy argument.";
          leaf type {
            type identityref {
              base argument-type;
            }
            description
              "Identifies the argument type.";
          }
          choice argument-choice {
            description
              "Argument formation options, depending on the policy
               type.";
            case policy-constant {
              leaf constant {
                type string;
                description
                  "The constant value of the policy argument.";
              }
            }
            case policy-variable {
              leaf policy-variable {
                type leafref {
                  path "/gnca/policy-variables/"
                    + "policy-variable/name";
                }
                description
                  "A reference to a global policy variable, which
                   is shared by all ECA scripts.";
              }
            }
            case local-policy-variable {
              leaf local-policy-variable {
                type leafref {
                  path "/gnca/ecas/eca/policy-variable/name";
                }
                description
                  "A reference to a local policy variable, which
                   is kept within an ECA instance, and appears/
                   disappears with start/stop of the ECA execution.";
              }
            }
            case xpath {
              leaf xpath {
                type string;
                description
                  "An XPath string, referencing the data in the
                   datastore.";
              }
            }
          }
        }
      } // policy-argument

      grouping action-element-attributes {
        description
          "Grouping of action element attributes.";
        leaf action-type {
          type identityref {
            base action-type;
          }
          description
            "Identifies the action type.";
        }
        choice action-operation {
          description
            "The operation choices that an ECA Action can take.";
          case action {
            leaf action-name {
              type leafref {
                path "/gnca/actions/action/name";
              }
              description
                "The operation is to execute a configured ECA Action.";
            }
          } // action
          case content-moving {
            container content-moving {
              description
                "The operation is to move contents between two policy
                 arguments.";
              leaf content-moving-type {
                type identityref {
                  base content-moving-operation-type;
                }
                description
                  "The type of moving operation, which can be copy,
                   iterate (copy a list of elements one by one), or
                   insert.";
              }
              container src {
                description
                  "The source policy argment.";
                uses policy-argument;
              }
              container dst {
                description
                  "The destination policy argument.";
                uses policy-argument;
              }
            }
          } // content-moving
          case function-call {
            container function-call {
              description
                "The operation is to call a function, which is of one of
```

```
                a few basic predefined types, such as plus, minus,
                multiply, devide, or remainder.";
             leaf function-type {
               type identityref {
                 base function-type;
               }
               description
                 "One of the predefined basic function types, such as
                  plus, minus, multiply, devide, or remainder.";
                     }
             container src {
               description
                 "The source policy argument.";
               uses policy-argument;
             }
             container dst {
               description
                 "The distination policy argument.";
               uses policy-argument;
             }
           }
         } // function-call
         case rpc-operation {
           container rpc-operation {
             description
               "The operation is to call an RPC, which is defined by
                a YANG module supported by the server.";
             leaf name {
               type string;
               description
                 "The name of the YANG RPC or YANG action to be
                  called.";
             }
             leaf nc-action-xpath {
               type string;
               description
                 "The location where the YANG action is defined.
                  This is used if and only if a YANG action is called.
                  This leaf is not set when a YANG RPC is called.";
             }
             list policy-variable {
               key name;
               description
                 "A list of policy arguments used as the input or output
                  parameters passed to the RPC.";
               leaf name {
                 type string;
                 description
```

```
                   "A string name used as the list key to form a list
                    of policy arguments.";
               }
               uses policy-argument;
             }
           }
         } // rpc-operation
         case notify-operation {
           container notify-operation {
             description
               "The operation is to send a YANG notification.";
             leaf name {
               type string;
               description
                 "Name of the subscribed YANG notification.";
             }
             list policy-variable {
               key name;
               description
                 "A list of policy arguments carried in the notification
                  message.";
               leaf name {
                 type string;
                 description
                   "A string name used as the list key to form a list
                    of policy arguments.";
               }
               uses policy-argument;
             }
           }
         } // notify-operation
       }
     } // action-element-attributes

     /*
      * Data nodes
      */
     container gnca {
       description
         "Top level container for Generalized Network Control Automation
          (GNCA).";

       // policy-variables
       container policy-variables {
         description
           "Container of global Policy Variables (PVs).";
         list policy-variable {
           key name;
           description
             "A list of global Policy Variables (PVs), with a string
              name as the entry key.";
           uses policy-variable-attributes;
         }
       } // policy-variables

       container conditions {
         description
           "Container of ECA Conditions.";
         list condition {
           key name;
           description
             "A list of ECA Conditions.";
           leaf name {
             type string;
             description
               "A string name to uniquely identify an ECA Condition
                globally.";
           }
           choice expression-choice {
             description
               "The choices of expression format to specify a condition,
                which can be either a XPath string or a YANG logical
                operation structure.";
             case logical-operation {
               leaf logical-operation-type {
                 type identityref {
                   base logical-operation-type;
                 }
                 description
                   "The logical operation type used to combine the
                    results from the list comparison-operation and the
                    list sub-condition, defined below.";
               }
               list comparison-operation {
                 key name;
                 description
                   "A list of comparison oprations, each of them defines
                    a comparison in the form of <arg1><relation><arg2>,
                    where <arg1> and <arg2> are policy arguments, while
                    <relation> is the comparison-type, which can be
                    ==, !=, >, <, >=, <=";
                 leaf name {
                   type string;
                   description
                     "A string name to uniquely identify a comparison
                      operation.";
                 }
                 leaf comparision-type {
                   type identityref {
                     base comparison-type;
                   }
                   description
                     "The comparison operation applied to the two
                      arguments arg1 and arg2 defined blow.";
                 }
                 container arg1 {
                   description
                     "The policy argument used as the first parameter of
                      the comparison opration.
                      A policy argument represents either a constant, PV
                      or data store value pointed by XPath.";
                   uses policy-argument;
                 }
                 container arg2 {
                   description
                     "The policy argument used as the secone parameter
                      of the comparison opration.
                      A policy argument represents either a constant, PV
                      or data store value pointed by XPath.";
                   uses policy-argument;
                 }
               }
               list sub-condition {
                 key name;
                 description
                   "A list of sub conditions applied by the
                    logical-operation-type. This list of sub conditions
                    provides the capability of hierarchically combining
                    conditions.";
                 leaf name {
                   type leafref {
                     path "/gnca/conditions/condition/name";
                   }
                   description
                     "A reference to a defined condition, which is used
                      as a sub-condition for the logical operation at
                      this hierarchy level.";
                 }
               } // sub-condition
             } // logical-operation
             case xpath {
               leaf condition-xpath {
                 type string;
                 description
```

```
                   "A XPath string, representing a logical expression,
                    which can contain comparisons of datastore values
                    and logical operations in the XPath format.";
               }
             } // xpath
           } // expression-choice
         } // condition
       } // conditions

       container actions {
         description
           "Container of ECA Actions.";
         list action {
           key name;
           description
             "A list of ECA Actions.";
           leaf name {
             type string;
             description
               "A string name to uniquely identify an ECA Action
                globally.";
           }

           list action-element {
             key name;
             description
               "A list of elements contained in an ECA Action. ";
             leaf name {
               type string;
               description
                 "A string name to uniquely identify the action element
                  within the scope of an ECA action.";
             }
             uses action-element-attributes;
           }

           container time-schedule {
             description
               "Specifying the time schedule to execute this ECA
                Action.
                If not specified, the ECA Action is executed immediately
                when it is called.";
             leaf start {
               type yang:date-and-time;
               description
                 "The start time of the ECA Action.
                  If not specified, the ECA Action is executed
                  immediately when it is called.";
```

```
            }
            leaf repeat-interval {
              type string {
                pattern
                  '(R\d*/)?P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?'
                  + '(\d+M)?(\d+S)?';
              }
              description
                "The repeat interval to execute this ECA Action.
                 The repeat interval is a string in ISO 8601 format,
                 representing a delay duration or a repeated delay
                 duration.
                 If not specified, the ECA Action is executed without
                 delay and without repetition.";
            }
          } // time-schedule
        }
      } // actions

      container ecas {
        description
          "Container of ECAs.";
        list eca {
          key name;
          description
            "A lis of ECAs";
          leaf name {
            type string;
            description
              "A string name to uniquely identify an ECA globally.";
          }
          leaf event-name {
            type string;
            mandatory true;
            description
              "The name of an event that triggers the execution of
               this ECA.";
          }

          list policy-variable {
            key name;
            description
              "A list of ECA local Policy Variables (PVs), with a
               string name as the entry key.";
            uses policy-variable-attributes;
            leaf is-static {
              type boolean;
              description
```

```
                 "'true' if the PV is static; 'false' if the PV is
                  dynamic.
                  A dynamic PV appears/disappears with the start/stop
                  of the ECA execution; a static PV exists as long as
                  the ECA is configured.";
             }
           }

           list condition-action {
             key name;
             description
               "A list of Condition-Actions, which are configured
                conditions each with associated actions to be executed
                if the condition is evaluated to TRUE";
             leaf name {
               type string;
               description
                 "A string name uniquely identify a Condition-Action
                  within this ECA.";
             }
             leaf condition {
               type leafref {
                 path "/gnca/conditions/condition/name";
               }
               description
                 "The reference to a configured condition.";
             }
             leaf action {
               type leafref {
                 path "/gnca/actions/action/name";
               }
               description
                 "The reference to a configured action.";
             }
           } // condition-action

           list cleanup-condition-action {
             key name;
             description
               "A list of Condition-Actions, which are configured
                conditions each with associated actions to be executed
                if the condition is evaluated to TRUE.
                This is the exception handler of this ECA, and is
                evaluated and executed in case any Action from the
                normal Condition-Action list was attempted and rejected
                by the server.";
             leaf name {
               type string;
               description
                 "A string name uniquely identify a Condition-Action
                  within this ECA.";
             }
             leaf condition {
               type leafref {
                 path "/gnca/conditions/condition/name";
               }
               description
                 "The reference to a configured condition.";
             }
             leaf action {
               type leafref {
                 path "/gnca/actions/action/name";
               }
               description
                 "The reference to a configured action.";
             }
           } // cleanup-condition-action

           action start {
             description
               "Start to execute this ECA.";
           }
           action stop {
             description
               "Stop the execution of this ECA.";
           }
           action pause {
             description
               "Pause the execution of this ECA.";
           }
           action resume {
             description
               "Resume the execution of this ECA.";
           }
           action next-action {
             description
               "Resume the execution of this ECA to complete the next
                action.";
           }
           list execution {
             key id;
             config false;
             description
               "A list of executions that this ECA has completed,
                are currently running, and will start in the scheduled
                future.";
             leaf id {
               type uint32;
               description
                 "The ID to uniquely identify an execution of the ECA.";
             }
             leaf oper-status {
               type enumeration {
                 enum completed {
                   description "Completed with no error.";
                 }
                 enum running {
                   description "Currently with no error.";
                 }
                 enum sleeping {
                   description "Sleeping because of time schedule.";
                 }
                 enum paused {
                   description "Paused by the operator.";
                 }
                 enum stoped {
                   description "Stopped by the operator.";
                 }
                 enum failed {
                   description "Failed with errors.";
                 }
                 enum error-handling {
                   description
                     "Asking the operator to handle an error.";
                 }
               }
               description
                 "The running status of the execution.";
             }
             leaf start-time {
               type yang:date-and-time;
               description
                 "The time when the ECA started.";
             }
             leaf stop-time {
               type yang:date-and-time;
               description
                 "The time when the ECA completed or stopped.";
             }
             leaf next-scheduled-time {
               type yang:date-and-time;
               description
                 "The next time when the ECA is scheduled to resume.";
             }
             leaf last-condition-action {
               type leafref {
                 path "../../condition-action/name";
               }
               description
                 "The reference to a condition-action last executed
                  or being executed.";
             }
             leaf last-condition {
               type leafref {
                 path "../../condition-action/condition";
               }
               description
                 "The reference to a condition last executed or being
                  executed.";
             }
             leaf last-action {
               type leafref {
                 path "../../condition-action/action";
               }
               description
                 "The reference to aa action last executed or being
                  executed.";
             }
             leaf last-cleanup-condition-action {
               type leafref {
                 path "../../cleanup-condition-action/name";
               }
               description
                 "The reference to a cleanup-condition-action last
                  executed or being executed.";
             }
           }
         }
       } // ecas

       container eca-scripts {
         description
           "Container of ECA Scripts.";
         list eca-script {
           key script-name;
           description
             "A list of ECA Script.";
           leaf script-name {
             type string;
             description
               "A string name to uniquely identify an ECA Script.";
           }
           list eca {
             key eca-name;
             description
               "A list of ECAs contained in this ECA Script.";
             leaf eca-name {
               type leafref {
                 path "/gnca/ecas/eca/name";
               }
               description
                 "The reference to a configured ECA.";
             }
           }
         }
       } // eca-scripts

       leaf running-script {
         type leafref {
           path "/gnca/eca-scripts/eca-script/script-name";
         }
         description
           "The reference to the ECA script that is currently running.";
       }
     }
   }
   <CODE ENDS>
```

8. IANA Considerations

TBD.

9. Security Considerations

TBD.

10. Acknowledgements

The authors would like to thank Joel Halpern and Robert Wilton for their helpful comments and valuable contributions.

11. References

11.1. Normative References

[I-D.clemm-netconf-push-smart-filters-ps]
Clemm, A., Voit, E., Liu, X., Bryskin, I., Zhou, T., Zheng, G., and H. Birkholz, "Smart filters for Push Updates - Problem Statement", draft-clemm-netconf-push-smart-filters-ps-00 (work in progress), October 2017.

[I-D.ietf-teas-yang-te-topo]
Liu, X., Bryskin, I., Beeram, V., Saad, T., Shah, H., and O. Dios, "YANG Data Model for Traffic Engineering (TE) Topologies", draft-ietf-teas-yang-te-topo-14 (work in progress), February 2018.

11.2. Informative References

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-09 (work in progress), January 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-14 (work in progress), February 2018.

Authors’ Addresses

Igor Bryskin
Huawei Technologies

EMail: Igor.Bryskin@huawei.com

Xufeng Liu
Jabil

EMail: Xufeng_Liu@jabil.com

Alexander Clemm
Huawei

EMail: ludwig@clemm.org

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Tianran Zhou
Huawei

EMail: zhoutianran@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2018

A. Clemm
Y. Qu
Huawei
J. Tantsura
Nuage Networks

February 23, 2018

Discrepancy detection between NMDA datastores
draft-clemm-netconf-nmda-diff-02

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 will expire on August 27, 2018.

Copyright Notice

Table of Contents

1. Introduction

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>, 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 discepancies. 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 dow 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. Values for patch-id and edit-id are generated by the server.

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)?
          |  |  +--:(subtree-filter)
          |  |  |  +---w subtree-filter?   <anydata>
          |  |  +--:(xpath-filter)
          |  |     +---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@2018-02-23.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 2018-02-23 {
    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, with
                   a value for patch-id and values for edit-id generated
                   by the server.  If there are no differences, an
```

```
                  empty container is returned.";
      }
    }
  }
}
<CODE ENDS>
```

6. Example

The following example compares the difference between <operational> and <intended> in any elements of list "instance" contained under "ospf", as defined in data module [I-D.draft-ietf-ospf-yang].

RPC request:

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <compare>
    <source>operational</source>
    <target>intended</target>
    <filter-spec>
      <xpath-filter>/ospf:ospf/ospf:instance/</xpath-filter>
    </filter-spec>
    <dampening>500</dampening>
  </compare>
</rpc>
```

RPC reply when a difference is detected:

```
     <rpc-reply
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       message-id="101">
       <differences>
         <yang-patch>
           <patch-id>p101</patch-id>
           <edit>
             <edit-id>1</edit-id>
             <operation>replace</operation>
             <target>
               /ospf:ospf/ospf:instance[ospf:af='1']/ospf:areas/
                 ospf:area[ospf:area-id='1']/ospf:default-cost
             </target>
             <value>
               <default-cost>128</default-cost>
             </value>
           </edit>
         </yang-patch>
       </differences>
     </rpc-reply>
```

RPC reply when no difference is detected:

```
      <rpc-reply
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
          message-id="101">
            <differences/>
      </rpc-reply>
```

7. IANA Considerations

7.1. 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:

URI: urn:ietf:params:xml:ns:yang:ietf-nmda-compare

Registrant Contact: The IESG.

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

7.2. 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

namespace: urn:ietf:params:xml:ns:yang:ietf-nmda-compare

prefix: cp

reference: RFC XXXX

8. 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.

9. Acknowledgments

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

10. Normative References

[I-D.draft-ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", I-D draft-ietf-netmod-yang-tree-diagrams, February 2018.

[I-D.draft-ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, J., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", I-D draft-ietf-ospf-yang, October 2017.

[NMDA] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", January 2018, <https://datatracker.ietf.org/doc/draft-ietf-netmod-revised-datastores/>.

[notif-sub]
Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom subscription to event notifications", January 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

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

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/info/rfc8072>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", February 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Authors’ Addresses

Alexander Clemm
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ludwig@clemm.org

Yingzhen Qu
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: yingzhen.qu@huawei.com

Jeff Tantsura
Nuage Networks

Email: jefftant.ietf@gmail.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

K. Watsen
Juniper Networks
October 30, 2017

YANG Data Model for a "Keystore" Mechanism
draft-ietf-netconf-keystore-04

Abstract

This document defines a YANG module called a "keystore", containing pinned certificates and pinned SSH host-keys. The module also defines a grouping for configuring public key pairs and a grouping for configuring certificates. The module also defines a notification that a system can use when one of its configured certificates is 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:

o "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:

o "2017-10-30" --> 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

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG [RFC7950] module for a system-level mechanism, herein called a "keystore". The keystore provides a centralized location for security sensitive data, as described below.

This module has the following characteristics:

- A 'grouping' for a public/private key pair, and an 'action' for requesting the system to generate a new private key.

- A 'grouping' for a list of certificates that might be associated with a public/private key pair, and an 'action' the requesting a system to generate a certificate signing request.

- An unordered list of pinned certificate sets, where each pinned certificate set contains an unordered list of pinned certificates. This structure enables a server to use specific sets of pinned certificates on a case-by-case basis. For instance, one set of pinned certificates might be used by an HTTPS-client when connecting to particular HTTPS servers, while another set of pinned certificates might be used by a server when authenticating client connections (e.g., certificate-based client authentication).

- An unordered list of pinned SSH host key sets, where each pinned SSH host key set contains an unordered list of pinned SSH host keys. This structure enables a server to use specific sets of pinned SSH host-keys on a case-by-case basis. For instance, SSH clients can be configured to use different sets of pinned SSH host keys when connecting to different SSH servers.

- A notification to indicate when a certificate is about to expire.

Special consideration has been given for systems that have Trusted Protection Modules (TPMs). These systems are unique in that the TPM must be directed to generate new keys (it is not possible to load a key into a TPM) and it is not possible to backup/restore the TPM's private keys as configuration.

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

1.1. 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.

2. Tree Diagram

The following tree diagram [I-D.ietf-netmod-yang-tree-diagrams] provides an overview of the data model for the "ietf-keystore" module.

```
module: ietf-keystore
    +--rw keystore
       +--rw pinned-certificates* [name]
       |  +--rw name                  string
       |  +--rw description?          string
       |  +--rw pinned-certificate* [name]
       |     +--rw name    string
       |     +--rw data    binary
       +--rw pinned-host-keys* [name]
          +--rw name               string
          +--rw description?       string
          +--rw pinned-host-key* [name]
             +--rw name    string
             +--rw data    binary

  notifications:
    +---n certificate-expiration
       +--ro certificate        instance-identifier
       +--ro expiration-date    yang:date-and-time

  grouping certificate-grouping
    +---- certificates
    |  +---- certificate* [name]
    |     +---- name?    string
    |     +---- value?   binary
    +---x generate-certificate-signing-request
       +---w input
       |  +---w subject       binary
       |  +---w attributes?   binary
       +--ro output
          +--ro certificate-signing-request    binary
  grouping private-key-grouping
    +---- algorithm?             identityref
    +---- private-key?           union
    +---- public-key?            binary
    +---x generate-private-key
       +---w input
          +---w algorithm    identityref
```

3. Example Usage

The following example illustrates what a configured keystore might look like.

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

  <!-- Manufacturer’s trust root CA certs -->
  <pinned-certificates>
    <name>manufacturers-root-ca-certs</name>
    <description>
      Certificates built into the device for authenticating
      manufacturer-signed objects, such as TLS server certificates,
      vouchers, etc..  Note, though listed here, these are not
      configurable; any attempt to do so will be denied.
    </description>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 1</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 2</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- pinned netconf/restconf client certificates -->
  <pinned-certificates>
    <name>explicitly-trusted-client-certs</name>
    <description>
      Specific client authentication certificates for explicitly
      trusted clients.  These are needed for client certificates
      that are not signed by a pinned CA.
    </description>
    <pinned-certificate>
      <name>George Jetson</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- pinned netconf/restconf server certificates -->
  <pinned-certificates>
    <name>explicitly-trusted-server-certs</name>
    <description>
      Specific server authentication certificates for explicitly
      trusted servers.  These are needed for server certificates
      that are not signed by a pinned CA.
    </description>
```

```
       <pinned-certificate>
         <name>Fred Flintstone</name>
         <data>base64encodedvalue==</data>
       </pinned-certificate>
     </pinned-certificates>

     <!-- trust anchors (CA certs) for authenticating clients -->
     <pinned-certificates>
       <name>deployment-specific-ca-certs</name>
       <description>
         Trust anchors (i.e. CA certs) that are used to authenticate
         client connections.  Clients are authenticated if their
         certificate has a chain of trust to one of these configured
         CA certificates.
       </description>
       <pinned-certificate>
         <name>ca.example.com</name>
         <data>base64encodedvalue==</data>
       </pinned-certificate>
     </pinned-certificates>

     <!-- trust anchors for random HTTPS servers on Internet -->
     <pinned-certificates>
       <name>common-ca-certs</name>
       <description>
         Trusted certificates to authenticate common HTTPS servers.
         These certificates are similar to those that might be
         shipped with a web browser.
       </description>
       <pinned-certificate>
         <name>ex-certificate-authority</name>
         <data>base64encodedvalue==</data>
       </pinned-certificate>
     </pinned-certificates>

     <!-- pinned SSH host keys -->
     <pinned-host-keys>
       <name>explicitly-trusted-ssh-host-keys</name>
       <description>
         Trusted SSH host keys used to authenticate SSH servers.
         These host keys would be analogous to those stored in
         a known_hosts file in OpenSSH.
       </description>
       <pinned-host-key>
         <name>corp-fw1</name>
         <data>base64encodedvalue==</data>
       </pinned-host-key>
     </pinned-host-keys>
```

```
   </keystore>
```

The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol.

[note: '\' line wrapping for formatting only]

```
   <notification
     xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
     <eventTime>2016-07-08T00:01:00Z</eventTime>
     <certificate-expiration
       xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
       <certificate xmlns:ks="urn:ietf:params:xml:ns:yang:ietf-keystore\
   ">
         /ks:keystore/ks:keys/ks:key[ks:name='ex-rsa-key']/ks:certifica\
   tes/ks:certificate[ks:name='ex-rsa-cert']
       </certificate>
       <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
     </certificate-expiration>
   </notification>
```

The following example module has been constructed to illustrate the groupings 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
   "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 uses the groupings defines the keystore draft
    for illustration.";

  revision "YYYY-MM-DD" {
    description
     "Initial version";
  }

  container key {
    uses ks:private-key-grouping;
    uses ks:certificate-grouping;
    description
      "A container of certificates, and an action to generate
       a certificate signing request.";
  }
}
```

The following example illustrates what a configured key might look like. This example uses the "ex-keystore-usage" module above.

```
[ note: '\' line wrapping for formatting only]

<key xmlns="http://example.com/ns/example-keystore-usage">
  <algorithm xmlns:ks="urn:ietf:params:xml:ns:yang:ietf-keystore">ks:\
secp521r1</algorithm>
  <private-key>base64encodedvalue==</private-key>
  <public-key>base64encodedvalue==</public-key>
  <certificates>
    <certificate>
      <name>domain certificate</name>
      <value>base64encodedvalue==</value>
    </certificate>
  </certificates>
 </key>
```

The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol. This example uses the "ex-keystore-usage" module above.

```
REQUEST
-------
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <key xmlns="http://example.com/ns/example-keystore-usage">
      <generate-certificate-signing-request>
        <subject>base64encodedvalue==</subject>
        <attributes>base64encodedvalue==</attributes>
      </generate-certificate-signing-request>
    </key>
  </action>
</rpc>

RESPONSE
--------
<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <certificate-signing-request
     xmlns="http://example.com/ns/example-keystore-usage">
     base64encodedvalue==
   </certificate-signing-request>
</rpc-reply>
```

The following example illustrates the "generate-private-key" action in use with the NETCONF protocol. This example uses the "ex-keystore-usage" module above.

```
   REQUEST
   -------
   [ note: '\' line wrapping for formatting only]

   <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0\
   ">
     <action xmlns="urn:ietf:params:xml:ns:yang:1">
       <key xmlns="http://example.com/ns/example-keystore-usage">
         <generate-private-key>
           <algorithm xmlns:ks="urn:ietf:params:xml:ns:yang:ietf-keysto\
   re">ks:secp521r1</algorithm>
         </generate-private-key>
       </key>
     </action>
   </rpc>

   RESPONSE
   --------
   <rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ok/>
   </rpc-reply>
```

4. YANG Module

This YANG module imports modules defined in [RFC6536] and [RFC6991]. This module uses data types defined in [RFC2315], [RFC2986], [RFC3447], [RFC4253], [RFC5280], [RFC5915], and [ITU.X690.1994]. This module uses algorithms defined in [RFC3447] and [RFC5480].

```
  <CODE BEGINS> file "ietf-keystore@2017-10-30.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-netconf-acm {
      prefix nacm;
      reference
        "RFC 6536: Network Configuration Protocol (NETCONF) Access
         Control Model";
    }

    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 a keystore to centralize management
      of security credentials.

      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 VVVV; see
      the RFC itself for full legal notices.";

    revision "2017-10-30" {
      description
       "Initial version";
      reference
       "RFC VVVV: YANG Data Model for a ’Keystore’ Mechanism";
    }

    // Identities

    identity key-algorithm {
      description
        "Base identity from which all key-algorithms are derived.";
    }

    identity rsa1024 {
      base key-algorithm;
      description
        "The RSA algorithm using a 1024-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa2048 {
      base key-algorithm;
      description
        "The RSA algorithm using a 2048-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa3072 {
      base key-algorithm;
      description
        "The RSA algorithm using a 3072-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa4096 {
      base key-algorithm;
      description
        "The RSA algorithm using a 4096-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa7680 {
      base key-algorithm;
      description
        "The RSA algorithm using a 7680-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa15360 {
      base key-algorithm;
      description
        "The RSA algorithm using a 15360-bit key.";
      reference
        "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                  RSA Cryptography Specifications Version 2.1.";
    }

    identity secp192r1 {
      base key-algorithm;
      description
        "The secp192r1 algorithm.";
      reference
        "RFC5480:
           Elliptic Curve Cryptography Subject Public Key Information.";
    }

    identity secp256r1 {
      base key-algorithm;
      description
        "The secp256r1 algorithm.";
      reference
        "RFC5480:
           Elliptic Curve Cryptography Subject Public Key Information.";
    }

    identity secp384r1 {
      base key-algorithm;
      description
        "The secp384r1 algorithm.";
      reference
        "RFC5480:
           Elliptic Curve Cryptography Subject Public Key Information.";
    }

    identity secp521r1 {
      base key-algorithm;
      description
        "The secp521r1 algorithm.";
      reference
        "RFC5480:
           Elliptic Curve Cryptography Subject Public Key Information.";
    }


    // typedefs

    typedef pinned-certificates {
      type leafref {
        path "/ks:keystore/ks:pinned-certificates/ks:name";
      }
      description
        "This typedef enables importing modules to easily define a
         reference to pinned-certificates.  Use of this type also
         impacts the YANG tree diagram output.";
      reference
        "I-D.ietf-netmod-yang-tree-diagrams: YANG Tree Diagrams";
    }

    typedef pinned-host-keys {
      type leafref {
        path "/ks:keystore/ks:pinned-host-keys/ks:name";
      }
      description
        "This typedef enables importing modules to easily define a
         reference to pinned-host-keys.  Use of this type also
         impacts the YANG tree diagram output.";
      reference
        "I-D.ietf-netmod-yang-tree-diagrams: YANG Tree Diagrams";
    }


    // groupings

    grouping private-key-grouping {
      description
        "A private/public key pair, and an action to request the
         system to generate a private key.";
      leaf algorithm {
        type identityref {
          base "key-algorithm";
        }
        description
          "Identifies the key's algorithm.  More specifically, this
           leaf specifies how the 'private-key' and 'public-key'
           binary leafs are encoded.";
      }
      leaf private-key {
        nacm:default-deny-all;
        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).";
            }
          }
        }
        must "../algorithm";
        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
            [RFC3447], and an Elliptic Curve Cryptography (ECC) key
            is represented as ECPrivateKey as defined in [RFC5915]";
         reference
           "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                      RSA Cryptography Specifications Version 2.1.
            RFC 5915: Elliptic Curve Private Key Structure.";
       }
       leaf public-key {
         type binary;
         must "../algorithm";
         must "../private-key";
         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
            [RFC3447], and an Elliptic Curve Cryptography (ECC) key
            is represented using the ’publicKey’ described in
            [RFC5915]";
         reference
           "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                      RSA Cryptography Specifications Version 2.1.
            RFC 5915: Elliptic Curve Private Key Structure.";
       }
       action generate-private-key {
         description
           "Requests the device to generate a private key using the
            specified key algorithm.  This action is primarily to
            support cryptographic processors that must generate
            the private key themselves.  The resulting key is
            considered operational state and hence only present
            in the <operational>.";
         input {
           leaf algorithm {
             type identityref {
               base "key-algorithm";
             }
             mandatory true;
             description
               "The algorithm to be used when generating the key.";
           }
         }
       } // end generate-private-key
     }
```

```
    grouping certificate-grouping {
      description
        "A container of certificates, and an action to generate
         a certificate signing request.";
      container certificates {
        description
          "Certificates associated with this key.  More than one
           certificate supports, for instance, a TPM-protected
           key that has both IDevID and LDevID certificates
           associated.";
        list certificate {
          key name;
          description
            "A certificate for this private key.";
          leaf name {
            type string;
            description
              "An arbitrary name for the certificate.";
          }
          leaf value {
            type binary;
            description
             "A PKCS #7 SignedData structure, as specified by
              Section 9.1 in RFC 2315, containing just certificates
              (no content, signatures, or CRLs), encoded using ASN.1
              distinguished encoding rules (DER), as specified in
              ITU-T X.690.

              This structure contains the certificate itself as well
              as any intermediate certificates leading up to a trust
              anchor certificate.  The trust anchor certificate MAY
              be included as well.";
            reference
              "RFC 2315:
                 PKCS #7: Cryptographic Message Syntax Version 1.5.
               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).";
          }
        }
      }
      action generate-certificate-signing-request {
        description
          "Generates a certificate signing request structure for
           the associated private 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 from 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.";
            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).";
          }
          leaf attributes {
            type binary;
            description
             "The ’attributes’ field from 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.";
            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 4.1 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).";

          }
        }
      }
    }


    // protocol accessible nodes

    container keystore {
      nacm:default-deny-write;
      description
        "The keystore contains X.509 certificates and SSH host keys.";

      list pinned-certificates {
        key name;
        description
          "A list of pinned certificates.  These certificates can be
           used by a server to authenticate clients, or by clients to
           authenticate servers.   Each list of pinned certificates
           SHOULD be specific to a purpose, as the list as a whole
           may be referenced by other modules.  For instance, a
           NETCONF server's configuration might use a specific list
           of pinned certificates for when authenticating NETCONF
           client connections.";
        leaf name {
          type string;
          description
            "An arbitrary name for this list of pinned certificates.";
        }
        leaf description {
          type string;
          description
            "An arbitrary description for this list of pinned
             certificates.";
        }
        list pinned-certificate {
          key name;
          description
            "A pinned certificate.";
          leaf name {
            type string;
            description
              "An arbitrary name for this pinned certificate. The
               name must be unique across all lists of pinned
               certificates (not just this list) so that leafrefs
               from another module can resolve to unique values.";
          }
          leaf data {
            type binary;
            mandatory true;
            description
              "An X.509 v3 certificate structure as specified by RFC
               5280, Section 4 encoded using the ASN.1 distinguished
               encoding rules (DER), as specified in ITU-T X.690.";
            reference
              "RFC 5280:
                 Internet X.509 Public Key Infrastructure Certificate
                 and Certificate Revocation List (CRL) Profile.
               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).";
          }
        }
      }

      list pinned-host-keys {
        key name;
        description
          "A list of pinned host keys.  These pinned host-keys can
           be used by clients to authenticate SSH servers.  Each
           list of pinned host keys SHOULD be specific to a purpose,
           so the list as a whole may be referenced by other modules.
           For instance, a NETCONF client’s configuration might
           point to a specific list of pinned host keys for when
           authenticating specific SSH servers.";
        leaf name {
          type string;
          description
            "An arbitrary name for this list of pinned SSH host keys.";
        }
        leaf description {
          type string;
          description
```

```
            "An arbitrary description for this list of pinned SSH host
             keys.";
        }
        list pinned-host-key {
          key name;
          description
            "A pinned host key.";
          leaf name {
            type string;
            description
              "An arbitrary name for this pinned host-key. Must be
               unique across all lists of pinned host-keys (not just
               this list) so that a leafref to it from another module
               can resolve to unique values.";
          }
          leaf data {
            type binary;
            mandatory true;
            description
              "The binary public key data for this SSH key, as
               specified by RFC 4253, Section 6.6, i.e.:

                 string    certificate or public key format
                           identifier
                 byte[n]   key/certificate data.";
            reference
              "RFC 4253: The Secure Shell (SSH) Transport Layer
                         Protocol";
          }
        }
      }
    }

    notification certificate-expiration {
      description
        "A notification indicating that a 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.";
      leaf certificate {
        type instance-identifier;
        mandatory true;
        description
          "Identifies which certificate is expiring or is expired.";
      }
      leaf expiration-date {
        type yang:date-and-time;
        mandatory true;
        description
          "Identifies the expiration date on the certificate.";
      }
    }

  }
  <CODE ENDS>
```

5. 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. This being the case, the top-level node in this module is marked with the NACM value 'default-deny-write'.

/keystore/keys/key/private-key: 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 then 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:

/keystore/keys/key/private-key: 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. This being the case, this node is marked with the NACM value 'default-deny-all'.

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:

generate-certificate-signing-request: 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.

6. IANA Considerations

6.1. 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:

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

6.2. The YANG Module Names Registry

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

```
      name:         ietf-keystore
      namespace:    urn:ietf:params:xml:ns:yang:ietf-keystore
      prefix:       ks
      reference:    RFC VVVV
```

7. 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.

8. References

8.1. Normative References

[ITU.X690.1994]
International Telecommunications 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, 1994.

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

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <https://www.rfc-editor.org/info/rfc3447>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

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

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC5915] Turner, S. and D. Brown, "Elliptic Curve Private Key Structure", RFC 5915, DOI 10.17487/RFC5915, June 2010, <https://www.rfc-editor.org/info/rfc5915>.

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

8.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-02 (work in progress), October 2017.

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

[RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <https://www.rfc-editor.org/info/rfc4211>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <https://www.rfc-editor.org/info/rfc5914>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[Std-802.1AR-2009] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

Appendix A. Change Log

A.1. 00 to 01

- Replaced the ’certificate-chain’ structures with PKCS#7 structures. (Issue #1)

- Added ’private-key’ as a configurable data node, and removed the ’generate-private-key’ and ’load-private-key’ actions. (Issue #2)

- Moved ’user-auth-credentials’ to the ietf-ssh-client module. (Issues #4 and #5)

A.2. 01 to 02

- Added back ’generate-private-key’ action.

- Removed ’RESTRICTED’ enum from the ’private-key’ leaf type.

- Fixed up a few description statements.

A.3. 02 to 03

- Changed draft’s title.

- Added missing references.

- Collapsed sections and levels.

- Added RFC 8174 to Requirements Language Section.

- Renamed ’trusted-certificates’ to ’pinned-certificates’.

- Changed ’public-key’ from config false to config true.

- Switched ’host-key’ from OneAsymmetricKey to definition from RFC 4253.

A.4. 03 to 04

- Added typedefs around leafrefs to common keystore paths

- Now tree diagrams reference ietf-netmod-yang-tree-diagrams

- Removed Design Considerations section

- Moved key and certificate definitions from data tree to groupings

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: August 26, 2018

A. Gonzalez Prieto
VMware
E. Voit
Cisco Systems
A. Clemm
Huawei
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems

February 22, 2018

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-08

Abstract

This document provides a NETCONF binding to subscribed notifications and to 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

Copyright Notice

Table of Contents

1. Introduction

This document provides a binding for events streamed over the NETCONF protocol [RFC6241] as per [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 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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

3. Interleave Capability

To support multiple subscriptions on a single session, a NETCONF publisher MUST support the :interleave capability as defined in [RFC5277]. Such support MUST be indicated by the following capability: "urn:ietf:params:netconf:capability:interleave:1.0". Advertisement of both the :interleave capability and "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications" within a NETCONF capability exchange MUST indicate that a NETCONF publisher is able to receive, process, and respond to NETCONF requests and [I-D.draft-ietf-netconf-subscribed-notifications] subscription operations.

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

A publisher is allowed to concurrently support both [I-D.draft-ietf-netconf-subscribed-notifications] and [RFC5277]’s "create-subscription" RPC, however this support MUST NOT happen on a single transport NETCONF session. To accomplish this:

- A solution must reply with the [RFC6241] error "operation-not-supported" if a "create-subscription" RPC is received on a NETCONF session where at least one subscription is currently active.
- It is a prohibited to send updates or state change notifications for a configured subscription on a NETCONF session where the create-subscription RPC has successfully created a subscription.
- A "create-subscription" RPC MUST be rejected if a subscription is already active across that NETCONF transport session.

5. Mandatory XML, stream and datastore support

A NETCONF publisher MUST support XML encoding of RPCs and Notifications.

A NETCONF publisher supporting [I-D.draft-ietf-netconf-subscribed-notifications] MUST support the "NETCONF" event stream identified in that draft.

A NETCONF publisher supporting [I-D.ietf-netconf-yang-push] MUST support the operational state datastore as defined by [I.D.draft-ietf-netmod-revised-datastores].

6. Transport connectivity

6.1. Dynamic Subscriptions

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

6.2. Configured Subscriptions

For a configured subscription, there is no guarantee a transport session is currently in place with each associated receiver. In cases where a configured subscription has a receiver in the connecting state and the protocol configured as NETCONF, but no NETCONF transport session exists to that receiver, the publisher MUST initiate a transport session via NETCONF call home [RFC8071], section 4.1 to that receiver. Until NETCONF connectivity is established and a subscription-started state change notification is successfully sent, that receiver MUST remain in a status of either "connecting" or "timeout".

If the 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 assigned a "timeout" status.

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 a "timeout" status after a predetermined number of either failed call home attempts or NETCONF sessions remotely terminated by the receiver.

NETCONF Transport session connectivity SHOULD be verified via Section 4.1, step S7.

If an active NETCONF session is disconnected but the stop-time of a subscription has not been reached, the publisher MUST restart the call home process and return the receiver to the "connecting" state.

7. Notification Messages

Notification messages transported over NETCONF will be identical in format and content to those encoded using one-way operations defined within [RFC5277], section 4.

8. 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:

- "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 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 (YANG-Push) subscriptions. The tag to use depends on the RPC for which the error occurred. Applicable are identities with a base identity of "establish-subscription-error" (for error responses to an establish-subscription request), "modify- subscription-error" (for error responses to a modify-subscription request), "delete-subscription-error" (for error responses to a delete-subscription request), "resynch-subscription-error" (for error responses to resynch-subscription request), or "kill- subscription-error" (for error responses to a kill-subscription request), respectively.
- In case of error responses to an establish-subscription or modify-subscription request: optionally, "error-info" containing XML-encoded data with hints regarding parameter settings that might lead to successful requests in the future, per yang-data definitions "establish-subscription-error-datastore" (for error responses to an establish-subscription request) or "modify-subscription-error-datastore (for error responses to a modify-subscription request), respectively.

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

  In case of an rpc error as a result of a delete-subscription, or 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 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.

9. Security Considerations

Notification messages (including state change notifications) are never sent before the NETCONF capabilities exchange has completed.

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.

The NETCONF Authorization Control Model [rfc6536bis] SHOULD be used to control and restrict authorization of subscription configuration.

10. 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, Balazs Lengyel, Martin Bjorklund, Mahesh Jethanandani, Kent Watsen, and Guangying Zheng.

11. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-10 (work in progress), January 2018.

[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", February 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[I.D.draft-ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

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

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

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

[rfc6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Access Control Module", December 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-rfc6536bis/>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

Appendix A. Examples

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].

The following example illustrates the retrieval of the list of available event streams using the "get" operation.

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

Figure 1: Get streams request

After such a request, the NETCONF publisher returns a list of event streams available.

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.

```
      +------------+                 +-----------+
      | Subscriber |                 | Publisher |
      +------------+                 +-----------+
            |                              |
            |    Capability Exchange       |
            |<---------------------------->|
            |                              |
            |                              |
            |    establish-subscription    |
            |----------------------------->|  (a)
            | RPC Reply: OK, id = 22       |
            |<-----------------------------|  (b)
            |                              |
            | notification message (for 22)|
            |<-----------------------------|
            |                              |
            |                              |
            |    establish-subscription    |
            |----------------------------->|
            | RPC Reply: OK, id = 23       |
            |<-----------------------------|
            |                              |
            |                              |
            | notification message (for 22)|
            |<-----------------------------|
            | notification message (for 23)|
            |<-----------------------------|
            |                              |
```

Figure 2: Multiple subscriptions over a NETCONF session

To provide examples of the information being transported, example messages for interactions (a) and (b) in Figure 2 are detailed below:

```
  <rpc netconf: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>
        <name>NETCONF</name>
        <xpath-filter xmlns:ex="http://example.com/events">
             /ex:foo
        </xpath-filter>
      </stream>
      <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>
     dscp-unavailable
    </error-app-tag>
    <error-info>
     <establish-subscription-error-datastore
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      <dscp>
       100
      </dscp>
    </error-info>
   </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.

```
      +------------+                +-----------+
      | Subscriber |                | Publisher |
      +------------+                +-----------+
            |                              |
            | notification message (for 23)|
            |<-----------------------------|
            |                              |
            | modify-subscription (id = 23)|
            |----------------------------->|  (c)
            | RPC error (with hint)        |
            |<-----------------------------|  (d)
            |                              |
            | modify-subscription (id = 23)|
            |----------------------------->|
            | RPC Reply: OK                |
            |<-----------------------------|
            |                              |
            | notification message (for 23)|
            |<-----------------------------|
            |                              |
```

Figure 6: Interaction model for successful subscription modification

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.

```
<rpc message-id="303"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore>
      <yp:xpath-filter xmlns="http://example.com/datastore">
        /interfaces-state/interface/oper-status
      </yp:xpath-filter>
      <yp:periodic>
        <yp:period>500</yp:period>
      </yp:periodic>
    </yp:datastore>
    <identifier>
      23
    </identifier>
  </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:

```
   <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>
           period-unsupported
       </error-message>
       <error-info
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
          <modify-subscription-error-datastore>
            <period-hint>
                3000
            </period-hint>
          </modify-subscription-error-datastore>
       </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.

```
  <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:

```
   <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>
           no-such-subscription
       </error-app-tag>
     </rpc-error>
   </rpc-reply>
```

Figure 10: Unsuccessful delete subscription

A.3. 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 NETCONF client.

A.3.1. Creating Configured Subscriptions

For subscription creation, a NETCONF client may send:

```
<rpc message-id="201"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscriptions
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      <subscription>
        <identifier>22</identifier>
        <encoding>encode-xml</encoding>
        <stream>
          <name>NETCONF</name>
          <receiver>
            <address>1.2.3.4</address>
            <port>1234</port>
          </receiver>
        </stream>
      </subscription>
    </subscriptions>
  </edit-config>
</rpc>
```

Figure 11: 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:

```
<rpc-reply message-id="201"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
        <error-type>application</error-type>
        <error-tag>resource-denied</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">
            Temporarily the publisher cannot serve this
            subscription due to the current workload.
        </error-message>
    </rpc-error>
</rpc-reply>
```

Figure 12: Response to a failed configured subscription establishment

After a subscription has been created, NETCONF connectivity to each receiver’s IP address and port will be established if it does not already exist. This will be accomplished via [RFC8071].

The following figure shows the interaction model for the successful creation of a configured subscription.

```
    +----------+                 +-----------+     +---------+
    |Config Ops|                 | Publisher |     | 1.2.3.4 |
    +----------+                 +-----------+     +---------+
         |                            |                |
         |    Capability Exchange     |                |
         |<-------------------------->|                |
         |                            |                |
         |                            |                |
         |        Edit-config         |                |
         |--------------------------->|                |
         |       RPC Reply: OK        |                |
         |<---------------------------|                |
         |                            |   Call Home    |
         |                            |<-------------->|
         |                            |                |
         |                            |  subscription- |
         |                            |  started       |
         |                            |--------------->|
         |                            |                |
         |                            |  notification  |
         |                            |  message       |
         |                            |--------------->|
```

Figure 13: Interaction model for configured subscription establishment

A.3.2. 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 a adding a second receiver:

```
<rpc message-id="202"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscriptions
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      <subscription>
        <identifier>
          1922
        </identifier>
        <receiver>
          <address>
            1.2.3.5
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscriptions>
  </edit-config>
</rpc>
```

Figure 14: Modify configured subscription

If the request is accepted, the publisher will indicate success. The result is that the interaction model described in Figure 13 may be extended as follows.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     | 1.2.3.4 |  | 1.2.3.5 |
 +----------+                 +-----------+     +---------+  +---------+
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  started       |            |
       |                            |---------------------------->|
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |---------------------------->|
       |                            |                |            |
```

Figure 15: Interaction model for configured subscription modification

Note in the above that in the specific example above, modifying a configured subscription actually resulted in "subscription-started" notification. And because of an existing NETCONF session, no additional call home was needed. Also note that if the edit of the configuration had impacted the filter, a separate modify-subscription would have been required for the original receiver.

A.3.3. 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.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     | 1.2.3.4 |  | 1.2.3.5 |
 +----------+                 +-----------+     +---------+  +---------+
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |---------------------------->|
       |                            |                |            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  terminated    |            |
       |                            |--------------->|            |
       |                            |---------------------------->|
       |                            |                |            |
```

Figure 16: Interaction model for configured subscription deletion

A.4. Subscription State Notifications

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

A.4.1. subscription-started and subscription-modified

A "subscription-started" over NETCONF encoded in XML would look like:

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

Figure 17: subscription-started subscription state notification

The "subscription-modified" is identical to Figure 17, with just the word "started" being replaced by "modified".

A.4.2. subscription-completed, subscription-resumed, and replay-complete

A "subscription-completed" would look like:

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

Figure 18: subscription-completed notification in XML

The "subscription-resumed" and "replay-complete" are virtually identical, with "subscription-completed" simply being replaced by "subscription-resumed" and "replay-complete" in both encodings.

A.4.3. subscription-terminated and subscription-suspended

A "subscription-terminated" would look like:

```
  <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 19: 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. v07 to v08

- Tweaks and clarification on :interleave.

B.2. v06 to v07

- XML encoding and operational datastore mandatory.
- Error mechanisms and examples updated.

B.3. v05 to v06

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

B.4. v03 to v04

- Added additional detail to "configured subscriptions"
- Added interleave capability
- Adjusted terminology to that in draft-ietf-netconf-subscribed-notifications
- Corrected namespaces in examples

B.5. v01 to v03

- Text simplifications throughout
- v02 had no meaningful changes

B.6. v00 to v01

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

Authors' Addresses

Alberto Gonzalez Prieto
VMware

Email: agonzalezpri@vmware.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2020

E. Voit
T. Jenkins
Cisco Systems
H. Birkholz
Fraunhofer SIT
A. Bierman
YumaWorks
A. Clemm
Futurewei

November 15, 2019

Notification Message Headers and Bundles
draft-ietf-netconf-notification-messages-08

Abstract

This document defines a new notification message format. 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

Copyright Notice

Table of Contents

1. Introduction

Mechanisms to support subscription to event notifications have been defined in [RFC8639] and [RFC8641]. Work on those documents has shown that notifications described in [RFC7950] section 7.16 could benefit from transport independent headers. With such headers, communicating the following information to receiving applications can be done without explicit linkage to an underlying transport protocol:

- the time the notification was generated

- the time the notification was placed in a message and queued for transport

- an identifier for the process generating the notification

- signatures to verify authenticity

- a subscription id which allows a notification be correlated with a request for that notification

- multiple notifications bundled into one transportable message

- a message-id allowing a receiver to check for message loss/ reordering

The document describes information elements needed for the functions above. It also provides instances of YANG structures [I-D.draft-ietf-netmod-yang-data-ext] for sending messages containing one or more notifications 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 [RFC7950] Section 4.2.10. Publisher, receiver, subscription, and event occurrence time are defined in [RFC8639].

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 origination time where a notification was fully generated within the publisher.

- notification-id: Identifies an instance of an emitted notification to a receiver.

- 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.

A receiver which supporting this document MUST be able to handle receipt of either type of message from a publisher.

4.1. One Notification per Message

This section has been deleted from previous versions. It will be re-instated if NETCONF WG members are not comfortable with the efficiency of the solution which can encode many notifications per message, as 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 YANG structure below, and is fully 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.

```
       structure 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 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:

```
   <structure 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-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>
         <push-change-update xmlns=
           "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
           <datastore-changes-xml>
             <alpha xmlns="http://example.com/sample-data/1.0">
                <beta urn:ietf:params:xml:ns:netconf:base:1.0:
                    operation="delete"/>
             </alpha>
           </datastore-changes-xml>
         </push-change-update>
       </notification-contents>
       <notification-header>
             ...(notification header, contents, footer)...
       </notification-footer>
     </notifications>
   </structure>
```

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 which are placed on all notifications. An optional header is a publisher default if its identity is included within the 'additional-headers' leaf-list.

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 such application specific configuration is accomplished within the publisher is out-of-scope.

The set of headers selected and populated for any particular message is derived from the union of the mandatory headers and configured optional headers.

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

```
module: ietf-notification-messages
    +--rw additional-default-headers {publisher}?
       +--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:

```
<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 welcome here.

The mechanism described above assumes that a capability discovery phase happens before a subscription is started. This is not always the case. There is no guarantee that a capability exchange has taken place before the messages are emitted. A solution for this in the case of HTTP based transport could be that a receiver would have to reply "ok" and also return the client capabilities as part a response to the initiation of the POST.

7. YANG Module

```
<CODE BEGINS> file "ietf-notification-messages@2019-10-10.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-yang-structure-ext { prefix sx; }


     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
       messages carrying notifications with well-known header objects.";

     revision 2019-10-10 {
       description
         "Initial version.";

       reference
         "draft-ietf-netconf-notification-messages-08";
     }

    /*
     * 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
      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 signature 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. If access control based on a message's receiver may
      strip information from within the notification, this notification-id MUST
      allow the identification of the specific contents of notification as it
      exits the 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 a signature which can be used to prove
      the authenticity that some asserter validates over 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;
    }
    description
      "Type of header object which may be included somewhere within a
      message.";
  }
  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 notifications in a bundled-message to a
        specific receiver.";
    }
  }

  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.";
    }
    leaf yang-module {
      type yang:yang-identifier;
      description
        "Name of the YANG module supported by the publisher.";
    }
    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 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 encoded structures which can be sent to receivers
   */

   sx:structure message {
    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'.";
      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 within a YANG module.";
      }
      leaf-list additional-notification-headers {
         type optional-notification-header;
         description
           "The set of additional default headers which will be sent
           for a specific notification.";
      }
    }
  }
}
```

<CODE ENDS>

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-Nygaard, and Kent Watsen.

11. References

11.1. Normative References

[I-D.draft-ietf-netmod-yang-data-ext]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", draft-ietf-netmod-yang-data-ext (work in progress), July 2019.

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

11.2. Informative References

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v06 - v08

- Removed redundant container from message

- References and example updates

v05 - v06

- With SN and YP getting RFC numbers, revisiting this document.

- Changed yang-data to draft-ietf-netmod-yang-data-ext's 'structure'.

- Removed the ability to reference structures other than YANG notifications.

v04 - v05

- Revision before expiration. Awaiting closure of SN and YP prior to update.

v03 - v04

- Terminology tweaks.

- Revision before expiration. Awaiting closure of SN prior to update.

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)

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

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.

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 [RFC8639]'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

Tim Jenkins
Cisco Systems

Email: timjenki@cisco.com

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Alexander Clemm
Futurewei

Email: ludwig@clemm.org

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 13, 2019

E. Voit
R. Rahman
E. Nilsen-Nygaard
Cisco Systems
A. Clemm
Huawei
A. Bierman
YumaWorks

June 11, 2019

Dynamic subscription to YANG Events and Datastores over RESTCONF
draft-ietf-netconf-restconf-notif-15

Abstract

This document provides a RESTCONF binding to the dynamic subscription capability of both subscribed notifications and YANG-Push.

Status of This Memo

Copyright Notice

Table of Contents

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 dynamic subscriptions over RESTCONF [RFC8040]. Requirements for these mechanisms are captured in [RFC7923].

The streaming of notifications encapsulating the resulting information push is done via the mechanism described in section 6.3 of [RFC8040].

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 use the definitions from [I-D.draft-ietf-netconf-subscribed-notifications]: 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 Subscriptions

This section provides specifics on how to establish and maintain dynamic subscriptions over RESTCONF [RFC8040]. Subscribing to event streams is accomplished in this way via RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.4. The RPCs are done via RESTCONF POSTs. 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.

As described in [RFC8040] Section 6.3, a GET 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" RPC has been accepted. Therefore, the POST for the "establish-subscription" RPC replaces the GET request for the "location" leaf which is used in [RFC8040] to obtain the URI. The subscription URI will be determined and sent as part of the response to the "establish-subscription" RPC, and a subsequent GET 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 GET is received.

3.1. Transport Connectivity

For a dynamic subscription, where a RESTCONF session doesn’t already exist, a new RESTCONF session is initiated from the subscriber.

As stated in Section 2.1 of [RFC8040], a subscriber MUST establish the HTTP session over TLS [RFC8446] in order to secure the content in transit.

Without the involvement of additional protocols, HTTP sessions by themselves do not 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 use a TLS heartbeat [RFC6520], just from subscriber to publisher, to track HTTP session continuity.

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 the dynamic subscription by using the procedure described in Section 3.4.

3.2. Discovery

Subscribers can learn what event streams a RESTCONF server supports by querying the "streams" container of ietf-subscribed-notification.yang in [I-D.draft-ietf-netconf-subscribed-notifications]. Support for the "streams" container of ietf-restconf-monitoring.yang in [RFC8040] is not required. In the case when the RESTCONF binding specified by this document is used to convey the "streams" container from ietf-restconf-monitoring.yang (i.e., that feature is supported), any event streams contained therein are also expected to be present in the "streams" container of ietf-restconf-monitoring.yang.

Subscribers can learn what datastores a RESTCONF server supports by following Section 2 of [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 the 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 an appropriate error code, as shown below, transported in the HTTP response.

When an HTTP error code is returned, the RPC reply MUST include an "rpc-error" element per [RFC8040] Section 7.1 with the following parameter values:

o an "error-type" node of "application".

o an "error-tag" node with the value being a string that corresponds to an identity associated with the error. This "error-tag" will come from one of two places. Either it will correspond to the error identities within [I-D.draft-ietf-netconf-subscribed-notifications] section 2.4.6 for general subscription errors:

error identity	uses error-tag	HTTP Code
dscp-unavailable	invalid-value	400
encoding-unsupported	invalid-value	400
filter-unsupported	invalid-value	400
insufficient-resources	resource-denied	409
no-such-subscription	invalid-value	404
replay-unsupported	operation-not-supported	501

Or this "error-tag" will correspond to the error identities within [I-D.ietf-netconf-yang-push] Appendix A.1 for subscription errors specific to YANG datastores:

error identity	uses error-tag	HTTP Code
cant-exclude	operation-not-supported	501
datastore-not-subscribable	invalid-value	400
no-such-subscription-resync	invalid-value	404
on-change-unsupported	operation-not-supported	501
on-change-sync-unsupported	operation-not-supported	501
period-unsupported	invalid-value	400
update-too-big	too-big	400
sync-too-big	too-big	400
unchanging-selection	operation-failed	500

o 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:

RPC	select an identity with a base
establish-subscription	establish-subscription-error
modify-subscription	modify-subscription-error
delete-subscription	delete-subscription-error
kill-subscription	delete-subscription-error
resync-subscription	resync-subscription-error

Each error identity will be inserted as the "error-app-tag" using JSON encoding following the form <modulename>:<identityname>. An example of such a 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 "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 "resync-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" [RFC8040] does not need to be included with the "rpc-error" element, as subscription errors are generally associated with the choice of RPC input parameters.

3.4. Call Flow for Server-Sent Events

The call flow for Server-Sent Events (SSE) is defined in Figure 1. The logical connections denoted by (a) and (b) can be a TCP connection or an HTTP2 stream (if HTTP2 is used, multiple HTTP2 streams can be carried in one TCP connection). Requests to [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push] augmented RPCs are sent on a 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 7.

An HTTP GET is then sent on a separate logical connection (b) to the URI on the publisher. This signals the publisher to initiate the flow of notification messages which are sent in SSE [W3C-20150203] as a response to the GET. There cannot be two or more simultaneous GET requests on a subscription URI: any GET request received while there is a current GET request on the same URI MUST be rejected with HTTP error code 409.

As described in [RFC8040] Section 6.4, RESTCONF servers SHOULD NOT send the "event" or "id" fields in the SSE event notifications.

```
   +--------------+                             +--------------+
   |  Subscriber  |                             |   Publisher  |
   |              |                             |              |
   |    Logical   |                             |     Logical  |
   |  Connection  |                             |   Connection |
   |  (a)  (b)    |                             |    (a)  (b)  |
   +--------------+                             +--------------+
       | RESTCONF POST (RPC:establish-subscription)   |
       |--------------------------------------------->|
       |                          HTTP 200 OK (ID,URI)|
       |<---------------------------------------------|
       |    |HTTP GET (URI)                                |
       |    |--------------------------------------------->|
       |    |                                   HTTP 200 OK|
       |    |<---------------------------------------------|
       |    |                           SSE (notif-message)|
       |    |<---------------------------------------------|
       | RESTCONF POST (RPC:modify-subscription)      |    |
       |--------------------------------------------->|    |
       |    |                              HTTP 200 OK|    |
       |<---------------------------------------------|    |
       |    |                   SSE (subscription-modified)|
       |    |<------------------------------------------(c)|
       |    |                           SSE (notif-message)|
       |    |<---------------------------------------------|
       | RESTCONF POST (RPC:delete-subscription)      |    |
       |--------------------------------------------->|    |
       |    |                              HTTP 200 OK|    |
       |<---------------------------------------------|    |
       |    |                                         |    |
       |    |                                         |    |
       (a) (b)                                       (a)  (b)
```

Figure 1: Dynamic with server-sent events

Additional requirements for dynamic subscriptions over SSE 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 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 (b). This allows the receiver to know exactly when, within the stream of events, the new terms of the subscription have been applied to the notification messages. See arrow (c).

- In addition to any required access permissions (e.g., NACM), RPCs modify-subscription, resync-subscription and delete-subscription SHOULD only be allowed by the same RESTCONF username [RFC8040] which invoked establish-subscription. Such a restriction generally serves to preserve users' privacy, but exceptions might be made for administrators that may need to modify or delete other users' subscriptions.

- The kill-subscription RPC can be invoked by any RESTCONF username with the required administrative permissions.

A publisher MUST terminate a subscription in the following cases:

- Receipt of a "delete-subscription" or a "kill-subscription" RPC for that subscription.

- Loss of TLS heartbeat

A publisher MAY terminate a subscription at any time as stated in [I-D.draft-ietf-netconf-subscribed-notifications] Section 1.3

4. QoS Treatment

Qos treatment for event streams is described in [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.3. In addition, if HTTP2 is used, the publisher MUST:

- take the "weighting" leaf node in [I-D.draft-ietf-netconf-subscribed-notifications], and copy it into the HTTP2 stream weight, [RFC7540] section 5.3, and

- take any existing subscription "dependency", as specified by the "dependency" leaf node in [I-D.draft-ietf-netconf-subscribed-notifications], and use the HTTP2 stream for the parent subscription as the HTTP2 stream dependency, [RFC7540] section 5.3.1, of the dependent subscription.

- set the exclusive flag, [RFC7540] section 5.3.1, to 0.

For dynamic subscriptions with the same DSCP value to a specific publisher, it is recommended that the subscriber sends all URI GET requests on a common HTTP2 session (if HTTP2 is used). Conversely, a subscriber can not use a common HTTP2 session for subscriptions with different DSCP values.

5. Notification Messages

Notification messages transported over RESTCONF will be encoded according to [RFC8040], section 6.4.

6. YANG Tree

The YANG model defined in Section 7 has one leaf augmented into three places of [I-D.draft-ietf-netconf-subscribed-notifications].

```
module: ietf-restconf-subscribed-notifications
  augment /sn:establish-subscription/sn:output:
    +--ro uri?   inet:uri
  augment /sn:subscriptions/sn:subscription:
    +--ro uri?   inet:uri
  augment /sn:subscription-modified:
    +--ro uri?   inet:uri
```

7. YANG module

This module references [I-D.draft-ietf-netconf-subscribed-notifications].

```
<CODE BEGINS> file
   "ietf-restconf-subscribed-notifications@2019-01-11.yang"
module ietf-restconf-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:" +
    "ietf-restconf-subscribed-notifications";

  prefix rsn;

  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 RESTCONF as a supported transport for subscribed
     event notifications.

     Copyright (c) 2019 IETF Trust and the persons identified as authors
     of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or without
     modification, is permitted pursuant to, and subject to the license
     terms contained in, the Simplified BSD License set forth in Section
     4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
     (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 2019-01-11 {
    description
      "Initial version";
    reference
      "RFC XXXX: RESTCONF Transport for Event 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 RESTCONF specific parameters for a
       response to a publisher's subscription request.";
    uses uri;
  }

  augment "/sn:subscriptions/sn:subscription" {
    description
      "This augmentation allows RESTCONF specific parameters to be
       exposed for a subscription.";
    uses uri;
  }

  augment "/sn:subscription-modified" {
    description
      "This augmentation allows RESTCONF specific parameters to be
       included as part of the notification that a subscription has been
       modified.";
    uses uri;
  }
}
<CODE ENDS>
```

8. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-subscribed-
notifications
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-restconf-subscribed-notifications
Namespace: urn:ietf:params:xml:ns:yang:ietf-restconf-subscribed-
notifications
Prefix: rsn
Reference: RFC XXXX: RESTCONF Transport for Event Notifications

9. 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 [RFC8446].

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, i.e., the same RESTCONF [RFC8040] user credentials which invoked the corresponding "establish-subscription", has the ability to access this resource.

The subscription URI is implementation specific and is encrypted via the use of TLS. Therefore, even if an attacker succeeds in guessing the subscription URI, a RESTCONF username [RFC8040] with the required administrative permissions must be used to be able to access or modify that subscription. It is recommended that the subscription URI values not be easily predictable.

The access permission considerations for the RPCs modify-subscription, resync-subscription, delete-subscription and kill-subscription are described in Section 3.4.

If a buggy or compromised RESTCONF subscriber sends a number of "establish-subscription" requests, then these subscriptions accumulate and may use up system resources. In such a situation, the publisher MAY also suspend or terminate a subset of the active subscriptions from that RESTCONF subscriber in order to reclaim resources and preserve normal operation for the other subscriptions.

10. 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, Guangying Zheng, Martin Bjorklund, Qin Wu and Robert Wilton.

11. References

11.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-21 (work in progress), January 2019.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-20 (work in progress), October 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

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

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

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <https://www.rfc-editor.org/info/rfc6520>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[W3C-20150203] Hickson, I., "Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <https://www.w3.org/TR/2015/REC-eventsource-20150203/>.

11.2. Informative References

[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-nmda-restconf] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", April 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-nmda-restconf/>.

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

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[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>.

[RFC8347] Liu, X., Ed., Kyparlis, A., Parikh, R., Lindem, A., and M. Zhang, "A YANG Data Model for the Virtual Router Redundancy Protocol (VRRP)", RFC 8347, DOI 10.17487/RFC8347, March 2018, <https://www.rfc-editor.org/info/rfc8347>.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

Appendix A. 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.

The subscription URI values used in the examples in this section are purely illustrative, and are not indicative of the expected usage which is described in Section 9.

The DSCP values are only for example purposes and are all indicated in decimal since the encoding is JSON [RFC7951].

A.1. Dynamic Subscriptions

A.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 |
            |<------------------------------|  (b)
            |GET (URI#1)                    |
            |------------------------------>|  (c)
            | HTTP 200 OK,notif-mesg (id#22)|
            |<------------------------------|
            |                               |
            |                               |
            |establish-subscription         |
            |------------------------------>|
            |      HTTP 200 OK, id#23, URI#2|
            |<------------------------------|
            |GET (URI#2)                    |
            |------------------------------>|
            |                               |
            |                               |
            |             notif-mesg (id#22)|
            |<------------------------------|
            | HTTP 200 OK,notif-mesg (id#23)|
            |<------------------------------|
            |                               |
```

Figure 2: Multiple subscriptions over RESTCONF/HTTP

To provide examples of the information being transported, example messages for interactions in Figure 2 are detailed below:

```
POST /restconf/operations
     /ietf-subscribed-notifications:establish-subscription

{
   "ietf-subscribed-notifications:input": {
      "stream-xpath-filter": "/example-module:foo/",
      "stream": "NETCONF",
      "dscp": 10
   }
}
```

Figure 3: 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

{
   "id": 22,
   "uri": "https://example.com/restconf/subscriptions/22"
}
```

Figure 4: establish-subscription success (b)

Upon receipt of the successful response, the subscriber does a GET the provided URI to start the flow of notification messages. When the publisher receives this, the subscription is moved to the active state (c).

```
GET /restconf/subscriptions/22
```

Figure 5: establish-subscription subsequent POST

While not shown in Figure 2, 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 3 proved unacceptable, the publisher may have returned:

```
HTTP status code - 400

{ "ietf-restconf:errors" : {
    "error" : [
      {
        "error-type": "application",
        "error-tag": "invalid-value",
        "error-severity": "error",
        "error-app-tag":
            "ietf-subscribed-notifications:dscp-unavailable"
      }
    ]
  }
}
```

Figure 6: an unsuccessful establish subscription

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

A.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)
         |    HTTP 400 error (with hint)|
         |<-----------------------------|  (e)
         |                              |
         |modify-subscription (id#23)   |
         |----------------------------> |
         |                  HTTP 200 OK |
         |<-----------------------------|
         |                              |
         |            notif-mesg (id#23)|
         |<-----------------------------|
         |                              |
```

Figure 7: Interaction model for successful subscription modification

If the subscription being modified in Figure 7 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 8. 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
     /ietf-subscribed-notifications:modify-subscription

{
 "ietf-subscribed-notifications:input": {
    "id": 23,
    "ietf-yang-push:datastore-xpath-filter":
       "/example-module:foo/example-module:bar",
    "ietf-yang-push:periodic": {
       "ietf-yang-push:period": 500
    }
  }
}
```

Figure 8: 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 - 400

{ "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "invalid-value",
      "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 9: Modify subscription failure with Hint (e)

A.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
     /ietf-subscribed-notifications:delete-subscription

{
 "delete-subscription": {
    "id": "22"
 }
}
```

Figure 10: 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 11 shows a valid response for existing valid subscription identifier, but that subscription identifier was created on a different transport session:

```
HTTP status code - 404

{
  "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "invalid-value",
      "error-severity": "error",
      "error-app-tag":
         "ietf-subscribed-notifications:no-such-subscription"
    ]
  }
}
```

Figure 11: Unsuccessful delete subscription

A.2. Subscription State Notifications

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

A.2.1. subscription-modified

A "subscription-modified" encoded in JSON would look like:

```
{
  "ietf-restconf:notification" : {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-modified": {
      "id": 39,
      "uri": "https://example.com/restconf/subscriptions/22"
      "stream-xpath-filter": "/example-module:foo",
      "stream": {
         "ietf-netconf-subscribed-notifications" : "NETCONF"
      }
    }
  }
}
```

Figure 12: subscription-modified subscription state notification

A.2.2. 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": {
      "id": 39,
    }
  }
}
```

Figure 13: 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".

A.2.3. 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": {
      "id": 39,
      "error-id": "suspension-timeout"
    }
  }
}
```

Figure 14: subscription-terminated subscription state notification

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

A.3. Filter Example

This section provides an example which illustrate the method of filtering event record contents. The example is based on the YANG notification "vrrp-protocol-error-event" as defined per the ietf-vrrp.yang module within [RFC8347]. Event records based on this specification which are generated by the publisher might appear as:

```
data: {
data:   "ietf-restconf:notification" : {
data:     "eventTime" : "2018-09-14T08:22:33.44Z",
data:     "ietf-vrrp:vrrp-protocol-error-event" : {
data:       "protocol-error-reason" : "checksum-error"
data:     }
data:   }
data: }
```

Figure 15: RFC 8347 (VRRP) - Example Notification

Suppose a subscriber wanted to establish a subscription which only passes instances of event records where there is a "checksum-error" as part of a VRRP protocol event. Also assume the publisher places such event records into the NETCONF stream. To get a continuous series of matching event records, the subscriber might request the application of an XPath filter against the NETCONF stream. An "establish-subscription" RPC to meet this objective might be:

```
POST /restconf/operations
     /ietf-subscribed-notifications:establish-subscription
{
   "ietf-subscribed-notifications:input": {
      "stream": "NETCONF",
      "stream-xpath-filter":
        "/ietf-vrrp:vrrp-protocol-error-event[
          protocol-error-reason='checksum-error']/",
   }
}
```

Figure 16: Establishing a subscription error reason via XPath

For more examples of XPath filters, see [XPATH].

Suppose the "establish-subscription" in Figure 16 was accepted. And suppose later a subscriber decided they wanted to broaden this subscription cover to all VRRP protocol events (i.e., not just those with a "checksum error"). The subscriber might attempt to modify the subscription in a way which replaces the XPath filter with a subtree filter which sends all VRRP protocol events to a subscriber. Such a "modify-subscription" RPC might look like:

```
POST /restconf/operations
     /ietf-subscribed-notifications:modify-subscription
{
   "ietf-subscribed-notifications:input": {
      "stream": "NETCONF",
      "stream-subtree-filter": {
        "/ietf-vrrp:vrrp-protocol-error-event" : {}
      }
   }
}
```

Figure 17

For more examples of subtree filters, see [RFC6241], section 6.4.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v14 - v15

- Addressed review comments from Kent.

v13 - v14

- Addressed review comments from IESG.

v12 - v13

- Enhanced "error-tag" values based on SN review.

v11 - v12

- Added text in 3.2 for expected behavior when ietf-restconf-monitoring.yang is also supported.
- Added section 2 to the reference to draft-ietf-netconf-nmda-restconf.
- Replaced kill-subscription-error by delete-subscription-error in section 3.3.
- Clarified vertical lines (a) and (b) in Figure 1 of section 3.4
- Section 3.4, 3rd bullet after Figure 1, replaced "must" with "MUST".
- Modified text in section 3.4 regarding access to RPCs modify-subscription, resync-subscription, delete-subscription and kill-subscription.
- Section 4, first bullet for HTTP2: replaced dscp and priority with weighting and weight.
- Section 6, added YANG tree diagram and fixed description of the module.
- Section 7, fixed indentation of module description statement.
- Section 7, in YANG module changed year in copyright statement to 2019.
- Section 8, added text on how server protects access to the subscription URI.
- Fixed outdated references and removed unused references.
- Fixed the instances of line too long.
- Fixed example in Figure 3.

v10 - v11

- Per Kent's request, added name attribute to artwork which need to be extracted

v09 - v10

- Fixed typo for resync.
- Added text wrt RPC permissions and RESTCONF username.

v08 - v09

- Addressed comments received during WGLC.

v07 - v08

- Aligned with RESTCONF mechanism.
- YANG model: removed augment of subscription-started, added restconf transport.
- Tweaked Appendix A.1 to match draft-ietf-netconf-netconf-event-notifications-13.
- Added Appendix A.3 for filter example.

v06 - v07

- Removed configured subscriptions.
- Subscription identifier renamed to id.

v05 - v06

- JSON examples updated by Reshad.

v04 - v05

- Error mechanisms updated to match embedded RESTCONF mechanisms
- Restructured format and sections of document.
- Added a YANG data model for HTTP specific parameters.
- Mirrored the examples from the NETCONF transport draft to allow easy comparison.

v03 - v04

- Draft not fully synched to new version of subscribed-notifications yet.

- References updated

v02 - v03

- Event notification reframed to notification message.

- Tweaks to wording/capitalization/format.

v01 - v02

- 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.

- 3rd party subscriptions are out-of-scope.

- SSE only used with RESTCONF and HTTP1.1 dynamic subscriptions

- Timeframes for event tagging are self-defined.

- Clean-up of wording, references to terminology, section numbers.

v00 - v01

- 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

NETCONF E. Voit
Internet-Draft Cisco Systems
Intended status: Standards Track A. Clemm
Expires: August 27, 2018 Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
February 23, 2018

Custom Subscription to Event Streams
draft-ietf-netconf-subscribed-notifications-10

Abstract

This document defines capabilities and operations for the customized establishment of subscriptions upon a publisher’s event streams. Also defined are delivery mechanisms for instances of the resulting notification messages. Effectively this allows a subscriber to request and receive a continuous, custom feed of publisher generated information.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines capabilities and operations for the customized establishment of subscriptions upon system generated event streams. Effectively this enables a "subscribe then publish" capability where the customized information needs 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, protocols 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].

1.1. Motivation

There are various [RFC5277] limitations, many of which have been exposed in [RFC7923] which needed to be solved. Key capabilities supported by this document includes:

- multiple subscriptions on a single transport session
- support for dynamic and statically configured subscriptions
- modification of an existing subscription
- operational counters and instrumentation
- negotiation of subscription parameters (through the use of hints returned as part of declined subscription requests)
- state change notifications (e.g., publisher driven suspension, parameter modification)
- independence from transport protocol

1.2. 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].

Configured subscription: A subscription installed via a configuration interface which persists across reboots.

Dynamic subscription: A subscription agreed between subscriber and publisher created via an establish-subscription RPC.

Event: An occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.)

Event record: A set of information detailing an event.

NACM: NETCONF Access Control Model.

Notification message: A set of transport encapsulated information intended for a receiver indicating that one or more event(s) have occurred. A notification message may bundle multiple event records. This includes the bundling of multiple, independent RFC 7950 YANG notifications.

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.

Stream (also referred to as "event stream"): A continuous ordered set of events aggregated under some context.

Stream filter: Evaluation criteria which may be applied against event records within a stream. Event records pass the filter when specified criteria are met.

Subscribed event records: Event records which have met the terms of the subscription. This include terms (such as security checks) enforced by the publisher.

Subscriber: An entity able to request and negotiate a contract for the generation and push of event records from a publisher.

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.

1.3. Solution Overview

This document describes a transport agnostic mechanism for subscribing to and receiving content from a stream instantiated 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 RPC. If the publisher wants to serve this request, it accepts it, and then starts pushing notification messages. If the publisher does not

wish to serve it as requested, then an error response is returned. This response MAY include hints at subscription parameters which would have been accepted.

2. Configured subscriptions, which allow the management of subscriptions via a configuration interface so that a publisher can send notification messages to configured receiver(s). Support for this capability is optional.

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bounded by the transport session used to establish it. For connection-oriented stateful transport 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 running 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.

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.

The publisher MAY decide to 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:

- the data model in this document replaces the data model in [RFC5277].

- the RPC operations in this draft replaces the symmetrical operations of [RFC5277], section 4.

- the one way operation of [RFC5277] is still used. However this operation will no longer be required with the availability of [I.D.draft-ietf-netconf-notification-messages].

- the definition and contents of the NETCONF stream are identical between this document and [RFC5277].

- a publisher MAY implement both the data model and RPCs defined in [RFC5277] and this new document concurrently, in order to support old clients. However the use of both alternatives on a single transport session is prohibited.

2. Solution

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, b) how event records are defined/generated, and c) how event records are assigned to streams.

There is only one reserved event stream within this document: NETCONF. The NETCONF event stream contains all NETCONF XML event record information supported by the publisher, except for where it has been explicitly indicated that this the event record MUST be excluded from the NETCONF stream. The NETCONF stream will include individual YANG notifications as per [RFC7950] section 7.16. Each of these YANG notifications will be treated a distinct event record. Beyond the NETCONF stream, implementations are free to add 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 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.

If access control permissions are in use to secure publisher content, then for event records to be sent to a receiver, that receiver MUST be allowed access to all the event records on the stream. If subscriber permissions change during the lifecycle of a subscription and stream access is no longer permitted, then the subscription MUST be terminated.

2.2. Event Stream Filters

This document defines an extensible filtering mechanism. Two optional stream filtering syntaxes supported are [XPATH] and subtree [RFC6241]. A filter always removes a complete event record; a subset of information is never stripped from an event record.

If no stream filter is provided within a subscription, all event records on a stream are to be sent.

2.3. QoS

This document provides an optional feature describing QoS parameters. These parameters indicate the treatment of a subscription relative to other traffic between publisher and receiver. Included are:

- A "dscp" QoS marking to differentiate transport QoS behavior. Where provided, this marking MUST be stamped on notification messages.

- 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. Notification messages MUST NOT be sent prior to other notification messages containing update record(s) for the referenced subscription.

A subscription’s weighting MUST work identically to stream dependency weighting as described within RFC 7540, section 5.3.2.

A subscription’s dependency MUST work identically to stream dependency as described within [RFC7540], sections 5.3.1, 5.3.3, and 5.3.4. If a dependency is attempted via an RPC, but the referenced subscription does not exist, the dependency will be silently removed.

2.4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC, and are 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.

2.4.1. Dynamic Subscription State Model

Below is the publisher’s state machine for a dynamic subscription. 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 asserted subscription to be externally visible. States that are reflected in the YANG model appear in upper-case letters; in addition, start and end states are depicted to reflect subscription creation and deletion events.

```
                       .........
                       : start :
                       :.......:
                           |
                  establish-subscription
                           |
                           |   .------modify-subscription-------.
                           v   v                                ’
                     .-----------.                         .-----------.
          .--------. | receiver  |-subscription-suspended->| receiver  |
        modify-     ’|  ACTIVE   |                         | SUSPENDED |
        subscription |           |<--subscription-resumed--|           |
          ---------->’-----------’                         ’-----------’
                           |                                    |
                delete/kill-subscription                   delete/kill-
                           |                               subscription
                           v                                    |
                       .........                                |
                       :  end  :<-------------------------------’
                       :.......:
```

Receiver state for a dynamic subscription

Of interest in this state machine are the following:

- o Successful establish or modify RPCs put the subscription into an active state.
- o Failed modify RPCs will leave the subscription in its previous state, with no visible change to any streaming updates.

- A delete or kill RPC will end the subscription.

- The two state change notifications "subscription-suspended" and "subscription-resumed" are shown. These are under the control of a publisher. There are no direct controls over suspend and resume other than to attempt a modification of a subscription.

2.4.2. Establishing a Subscription

The "establish-subscription" operation allows a subscriber to request the creation of a subscription via RPC. It MUST be possible to support multiple establish subscription RPC requests made within the same transport session. And it MUST be possible to support the interleaving of RPC requests made on independent subscriptions.

The input parameters of the operation are:

- A stream name which identifies the targeted stream of events against which the subscription is applied.

- A stream filter which may reduce the set of event records pushed.

- 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. If the start-time is in the past, it indicates that this subscription is requesting a replay of previously generated information from the event stream. For more on replay, see Section 2.4.2.1.

If the publisher can satisfy the "establish-subscription" request, it provides an identifier for the subscription, and immediately starts streaming notification messages.

```
    +---x establish-subscription
       +---w input
       |  +---w encoding?                encoding
       |  +---w (target)
       |  |  +--:(stream)
       |  |     +---w (stream-filter)?
       |  |     |  +--:(by-reference)
       |  |     |  |  +---w stream-filter-ref        stream-filter-ref
       |  |     |  +--:(within-subscription)
       |  |     |     +---w (filter-spec)?
       |  |     |        +--:(stream-subtree-filter)
       |  |     |        |  +---w stream-subtree-filter?    {subtree}?
       |  |     |        +--:(stream-xpath-filter)
       |  |     |           +---w stream-xpath-filter?
       |  |     |                                 yang:xpath1.0 {xpath}?
       |  |     +---w stream?               stream-ref
       |  |     +---w replay-start-time?    yang:date-and-time {replay}?
       |  +---w stop-time?               yang:date-and-time
       |  +---w dscp?                    inet:dscp {qos}?
       |  +---w weighting?               uint8 {qos}?
       |  +---w dependency?              sn:subscription-id {qos}?
       +--ro output
          +--ro identifier                subscription-id
```

Figure 1: establish-subscription RPC

A publisher MAY reject this RPC for many reasons as described in Section 2.4.6. The contents of the resulting RPC error response MAY include one or more hints on alternative inputs which would have resulted in a successfully established subscription. Any such hints MUST be transported within a yang-data "establish-subscription-error-stream" container included within the RPC error response.

```
       yang-data establish-subscription-error-stream
          +--ro establish-subscription-error-stream
             +--ro reason?                   identityref
             +--ro filter-failure-hint?      string
             +--ro replay-start-time-hint?   yang:date-and-time
```

Figure 2: establish-subscription RPC yang-data

2.4.2.1. Replay Subscription

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 target event stream which meets the filter and timeframe criteria. These historical event records would then be followed by event records generated after the subscription has been established. All event records will be delivered in the order generated.

Replay is an optional feature which is dependent on an event stream supporting some form of logging. Replay puts no restrictions on the size or form of the log, or where it resides within the device.

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 content stored within the publisher's replay buffer, then the subscription MUST be rejected, and the leaf "replay-start-time-hint" MUST be set in the reply.

If a "stop-time" parameter is included, it MAY also be earlier than the current time and MUST be later than the "replay-start-time". The publisher MUST NOT accept a "replay-start-time" for a future time.

If the "replay-start-time" is later than any information stored in the replay buffer, then the publisher MUST send a "replay-completed" notification immediately after the "subscription-started" notification.

If a 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 availability of replay, subscribers can perform a get on "replay-log-creation-time" and "replay-log-aged-time". See Figure 10 for the tree 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 Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription previously established on that transport session via "establish-subscription". Dynamic subscriptions can be modified one or multiple 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.

Dynamic subscriptions established via RPC can only be modified via RPC using the same transport session used to establish that subscription. Subscriptions created by configuration operations cannot be modified via this RPC.

```
    +---x modify-subscription
       +---w input
          +---w identifier?            subscription-id
          +---w (target)
          |  +--:(stream)
          |     +---w (stream-filter)?
          |        +--:(by-reference)
          |        |  +---w stream-filter-ref        stream-filter-ref
          |        +--:(within-subscription)
          |           +---w (filter-spec)?
          |              +--:(stream-subtree-filter)
          |              |  +---w stream-subtree-filter?    {subtree}?
          |              +--:(stream-xpath-filter)
          |                 +---w stream-xpath-filter?
          |                                      yang:xpath1.0 {xpath}?
          +---w stop-time?             yang:date-and-time
```

Figure 3: modify-subscription RPC

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 an 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 a such a rejected RPC MAY include one or more hints on alternative inputs which would have resulted in a successfully modified subscription. These hints MUST be transported within a yang-data "modify-subscription-error-stream" container inserted into the RPC error response.

```
       yang-data modify-subscription-error-stream
          +--ro modify-subscription-error-stream
             +--ro reason?               identityref
             +--ro filter-failure-hint?   string
```

Figure 4: modify-subscription RPC yang-data

2.4.4. Deleting a Subscription

The "delete-subscription" operation permits canceling an existing subscription previously established on that transport session. 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 publisher rejects the request, the request has no impact whatsoever on any subscription.

```
    +---x delete-subscription
       +---w input
          +---w identifier    subscription-id
```

Figure 5: delete-subscription RPC

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 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. This operation MUST be secured so that only connections with sufficiently privileged access rights are able to invoke this RPC. A publisher MUST terminate any dynamic subscription identified by RPC request. An operator may find subscription identifiers which may be used with "kill-subscription" by searching for the IP address of a receiver within "subscriptions\subscription\receivers\receiver\address".

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.

The tree structure of "kill-subscription" is almost identical to "delete-subscription", with only the name of the RPC and yang-data changing.

2.4.6. RPC Failures

Whenever an RPC is unsuccessful, the publisher returns relevant error codes as part of the RPC error response. RPC error codes returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241] Appendix A, as well as subscription specific errors such as those 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.

There are elements of the RPC error mechanism which are transport independent. Specifically, references to specific identities within the YANG model MUST be returned as part of the error responses resulting from failed attempts at event stream subscription. Following are valid errors per RPC:

```
 establish-subscription         modify-subscription
 ----------------------         -------------------
 dscp-unavailable               filter-unsupported
 filter-unsupported             insufficient-resources
 history-unavailable            no-such-subscription
 insufficient-resources
 replay-unsupported

 delete-subscription            kill-subscription
 ----------------------         ----------------------
 no-such-subscription            no-such-subscription
```

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. yang-data establish-subscription-error-stream: 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. yang-data modify-subscription-error-stream: 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. yang-data delete-subscription-error: 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 a configuration interface. Configured subscriptions may be modified by any configuration client with the proper permissions. Subscriptions can be modified or terminated via the configuration interface at any point of their lifetime.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions:

- persistence across reboots,

- persistence even when transport is unavailable, and

- an ability to send notification messages to more than one receiver (note that the publisher does not provide information to a receiver about other receivers.)

Supporting configured subscriptions is optional and advertised using the "configured" feature.

In addition to subscription parameters available to dynamic subscriptions as described in Section 2.4.2, the following additional parameters are also available to configured subscriptions:

- One or more receiver IP addresses (and corresponding ports) intended as the destination for notification messages.

- Optional parameters to identify an egress interface, a host IP address, a VRF (as defined by the network instance name within [I-D.draft-ietf-rtgwg-ni-model]), or an IP address plus VRF out of which notification messages are to be pushed from the publisher. Where any of this info is not explicitly included, or where just the VRF is provided, notification messages MUST egress the publisher’s default interface towards that receiver.

2.5.1. Configured Subscription State Model

Below is the state machine for a configured subscription. States that are reflected in the YANG model appear in upper-case letters; in addition, start and end states are depicted to reflect configured subscription creation and deletion events. The creation or modification of a configured subscription initiates a publisher evaluation to determine if the subscription is valid or invalid. The publisher uses its own criteria in making this determination. If valid, the subscription becomes operational.

```
 .........
 : start :-.
 :.......: |
      create  .---modify-----.----------------------------------.
           |  |              |                                  |
           V  V          .-------.         .......         .---------.
  .----[evaluate]--no--->|INVALID|-delete->: end :<-delete-|CONCLUDED|
  |                      '-------'         :.....:         '---------'
  |----[evaluate]--no-.      ^                ^                 ^
  |        ^          |      |                |                 |
 yes       |          '->unsupportable      delete           stop-time
  |      modify         (subscription-   (subscription-   (subscription-
  |        |              terminated)      terminated)      concluded)
  |        |                 |                |                 |
  |   .---------------------------------------------------------------.
  '-->|                         VALID                                 |
      '---------------------------------------------------------------'
```

State model for a configured subscription.

A valid subscription may become invalid on one of two ways. First, it may be modified in a way which fails a re-evaluation. Second, the publisher itself might itself determine that the subscription in no longer supportable. In either case, a "subscription-terminated" notification is sent to any active or suspended receivers. A valid subscription may also transition to a concluded state if a configured stop time has been reached. In this case, a "subscription-concluded" is sent to any active or suspended receivers.

During any times a subscription is considered valid, a publisher will attempt to connect with all configured receivers and deliver notification messages. Below is the state machine for each receiver of a configured subscription. This receiver state machine itself is fully contained within the state machine of the configured subscription, and is only relevant when the configured subscription itself is determined to be valid.

```
      .----------------------------------------------------------------.
      |                         VALID                                  |
      |   .----------.                           .--------.            |
      |   | receiver |------------------timeout->|receiver|            |
      |   |CONNECTING|<------------------reset---|TIMEOUT |            |
      |   |          |<-transport---.            ’--------’            |
      |   ’----------’  loss,reset  |                                  |
      |       |           |         |                                  |
      | (subscription     |         |                                  |
      |  started)    .--------.     |                      .---------. |
      |       ’----->|        |     ’----------------------|         | |
      |              |receiver|-(subscription-suspended)-->|receiver | |
      |(subscription-| ACTIVE |                            |SUSPENDED| |
      |   modified)  |        |<-(subscription-resumed,----|         | |
      |        ’---->’--------’   subscription-modified)   ’---------’ |
      ’----------------------------------------------------------------’
```

Receiver state for a configured subscription

When a subscription first becomes valid, the operational state of each receiver is initialized to connecting. Individual receivers are moved to an active state when a "subscription-started" state change notification is successfully passed to that receiver. Configured receivers remain active if transport connectivity is not lost, and event records are not being dropped due to a publisher buffer overflow. A configured subscription’s receiver MUST be moved to connecting if transport connectivity is lost, or if the receiver is reset via configuration operations.

A configured subscription’s receiver MUST be moved to a suspended state if there is transport connectivity between the publisher and receiver, but notification messages are not being generated for that receiver. A configured subscription receiver MUST be returned to an active state from the suspended state when notification messages are again being generated and a receiver has successfully been sent a "subscription-resumed" or a "subscription-modified".

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.

The mechanisms described above is mirrored in the RPCs and YANG notifications within the document. It should be noted that these RPCs and YANG notifications have been designed to be extensible and allow subscriptions into targets other than event streams.

[I-D.ietf-netconf-yang-push] provides an example of such an extension.

2.5.2. Creating a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level "subscriptions" subtree. There are two key differences between the new RPCs defined in this document and configuration operations for subscription creation. Firstly, configuration operations install a subscription without question, while the RPCs are designed to the support negotiation and rejection of requests. Secondly, while the RPCs mandate that the subscriber establishing the subscription is the only receiver of the notification messages, configuration operations permit specifying receivers independent of any tracked subscriber. Because there is no explicit association with an existing transport session, configuration operations require additional parameters beyond those of dynamic subscriptions to indicate receivers, and possibly whether the notification messages need to come from a specific egress interface on the publisher.

After a subscription is successfully created, 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 at subscription creation using configuration operations over NETCONF, see Appendix A of [I-D.draft-ietf-netconf-netconf-event-notifications].

Note that 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 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.)

2.5.4. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations 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 Receiver

It is possible that a configured subscription to a receiver needs to be 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.

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 set up via RPC operations, 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, plus receiver IP address and port configured. In all cases, a single transport session MUST be able to support the interleaving of event records, RPCs, and state change notifications from independent subscriptions.

A notification message is sent to a receiver when an event record is able to traverse the specified filter criteria. This notification message MUST be encoded as one-way notification element of [RFC5277], Section 4. A subscription’s events MUST NOT be sent to a receiver until after a corresponding RPC response or state-change notification has been passed receiver indicating that events should be expected. The following example within [RFC7950] section 7.16.3 is an example of a compliant message:

```
<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 6: subscribed notification message

This [RFC5277] section 4 one-way operation has the drawback of not including useful header information such as a subscription identifier. When using this mechanism, it is left up to implementations or augmentations to this document to determine which event records belong to which subscription.

These drawbacks, along with other useful common headers and the ability to bundle multiple event records together is being explored within [I.D.draft-ietf-netconf-notification-messages]. When the notification-messages is supported, this document will be updated to indicate support.

2.7. Subscription State Notifications

In addition to subscribed event records, a publisher MUST send subscription state notifications to indicate to receivers that an event related to the subscription management has occurred.

Subscription state notifications are unlike other notifications which might be found in the event stream. They cannot be filtered out, and they are delivered only to directly impacted receiver(s) 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 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) addressing 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 will still provide the contents of that referenced filter under the "within-subscription" subtree.

Note that for dynamic subscriptions, no "subscription-started" notifications are ever sent.

```
+---n subscription-started {configured}?
   +--ro identifier              subscription-id
   +--ro protocol                transport {configured}?
   +--ro encoding                encoding
   +--ro (target)
   |  +--:(stream)
   |     +--ro (stream-filter)?
   |     |  +--:(by-reference)
   |     |  |  +--ro stream-filter-ref        stream-filter-ref
   |     |  +--:(within-subscription)
   |     |     +--ro (filter-spec)?
   |     |        +--:(stream-subtree-filter)
   |     |        |  +--ro stream-subtree-filter?    {subtree}?
   |     |        +--:(stream-xpath-filter)
   |     |           +--ro stream-xpath-filter?   yang:xpath1.0 {xpath}?
   |     +--ro stream                   stream
   |     +--ro replay-start-time?       yang:date-and-time {replay}?
   +--ro stop-time?              yang:date-and-time
   +--ro dscp?                   inet:dscp {qos}?
   +--ro weighting?              uint8 {qos}?
   +--ro dependency?             sn:subscription-id {qos}?
```

Figure 7: subscription-started notification

2.7.2. subscription-modified

This notification indicates that a subscription has been modified by configuration operations. The same parameters of "subscription-started" are provided via this notification. As a result, the tree structure of "subscription-modified" is almost identical to "subscription-started", with only the name of the notification changing.

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.

- o First, 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.

- o Second, for dynamic subscriptions, there is one and only one time this notification may be sent. A "subscription-modified" state change notifications MUST be sent if the contents of a filter identified by a "stream-filter-ref" has changed.

2.7.3. subscription-terminated

The publisher MAY decide to terminate the pushing of subscribed event records to a receiver. This notification indicates that no further notification messages should be expected from the publisher. Such a decision may be made for two types of reasons. The first type of reason is that a subscription’s referenced objects are no longer accessible via the YANG model. Identities within the YANG model corresponding to such a loss include: "filter-unavailable", "no-such-subscription", and "stream-unavailable". The second reason is that a suspended subscription has exceeded some timeout. This condition is indicated via the identity "suspension-timeout". Publisher-driven terminations are always notified to all receivers.

```
    +---n subscription-terminated
       +--ro identifier    subscription-id
       +--ro reason        identityref
```

Figure 8: subscription-terminated notification

Note: subscribers themselves can terminate existing subscriptions established via a "delete-subscription" RPC. In such cases, no "subscription-terminated" state change notifications are sent. However if a "kill-subscription" RPC is sent, or some other event other than reaching the subscription’s stop time results in the end of a subscription, then this state change notification MUST be sent.

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 is this possible are "insufficient-resources" and "unsupportable-volume". No further notification will be sent until the subscription resumes or is terminated.

The tree structure of "subscription-suspended" is almost identical to "subscription-terminated", with only the name of the notification changing.

2.7.5. subscription-resumed

This indicates that a previously suspended subscription has been resumed under the unmodified terms previously in place. Subscribed event records generated after the generation of this state change notification will be sent.

```
    +---n subscription-resumed
       +--ro identifier    subscription-id
```

Figure 9: subscription-resumed notification

2.7.6. subscription-completed

This notification indicates that a subscription, which includes a "stop-time", has successfully finished passing event records upon the reaching of that time.

The tree structure of "subscription-completed" is almost identical to "subscription-resumed", with only the name of the notification changing.

2.7.7. replay-completed

This notification indicates that all of the event records prior to the current time have been sent. This includes new event records generated since the start of the subscription. This notification MUST NOT be sent for any other reason.

If subscription contains no "stop-time", or has a "stop-time" which 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.

The tree structure of "replay-completed" is almost identical to "subscription-resumed", with only the name of the notification changing.

2.8. Subscription Monitoring

Container "subscriptions" in the YANG module contains the state of all known subscriptions. This includes subscriptions that were established (and have not yet been deleted) using RPCs, as well as subscriptions that have been configured as part of configuration. Using the "get" operation with NETCONF, or subscribing to this information via [I-D.ietf-netconf-yang-push] allows the state of subscriptions and their connectivity to receivers to be monitored.

Each subscription is represented as a list element. The associated information includes an identifier for the subscription, receiver counter information, the state of the receiver (e.g., is currently active or suspended), as well as the various subscription parameters that are in effect. Leaf "configured-subscription-state" indicates that the subscription came into being via configuration, and the current state of the configured subscription.

Subscriptions that were established by RPC are removed from the list once they expire (reaching stop-time) or when they are terminated. Subscriptions that were established by configuration need to be deleted from the configuration by a configuration editing operation even if the stop time has been passed.

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", and "replay" MUST also be indicated if supported.

If a publisher supports this specification but not subscriptions via [RFC5277], the publisher MUST NOT advertise "urn:ietf:params:netconf:capability:notification:1.0". Even without this advertisement however, the publisher MUST support the one-way notification element of [RFC5277] Section 4.

3. YANG Data Model Trees

This section contains tree diagrams for top level YANG Data Node containers defined in Section 4. If you would rather see tree diagrams for 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. The list of event streams that are supported by the publisher and against which subscription is allowed may be acquired from the "streams" container within the YANG module.

```
 +--rw streams
    +--rw stream* [name]
       +--rw name                          stream
       +--rw description                   string
       +--rw replay-support?               empty {replay}?
       +--rw replay-log-creation-time?   yang:date-and-time {replay}?
       +--rw replay-log-aged-time?       yang:date-and-time {replay}?
```

Figure 10: Stream Container

3.2. Event Stream Filters Container

The "filters" container maintains a list of all subscription filters which persist outside the life-cycle of a single subscription. This enables pre-defined and validated filters which may be referenced and used 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?    {subtree}?
             +--:(stream-xpath-filter)
                +--rw stream-xpath-filter?     yang:xpath1.0 {xpath}?
```

Figure 11: Filter Container

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.

```
  +--ro subscriptions
     +--ro subscription* [identifier]
        +--ro identifier                       subscription-id
        +--ro configured-subscription-state?   enumeration {configured}?
        +--ro purpose?                         string {configured}?
        +--ro protocol                         transport {configured}?
        +--ro encoding                         encoding
        +--ro (target)
        |  +--:(stream)
        |     +--ro (stream-filter)?
        |     |  +--:(by-reference)
        |     |  |  +--ro stream-filter-ref          stream-filter-ref
        |     |  +--:(within-subscription)
        |     |     +--ro (filter-spec)?
        |     |        +--:(stream-subtree-filter)
        |     |        |  +--ro stream-subtree-filter?      {subtree}?
        |     |        +--:(stream-xpath-filter)
        |     |           +--ro stream-xpath-filter?
        |     |                                yang:xpath1.0 {xpath}?
        |     +--ro stream?                    stream-ref
        |     +--ro replay-start-time?    yang:date-and-time {replay}?
        +--ro stop-time?                     yang:date-and-time
        +--ro dscp?                          inet:dscp {qos}?
        +--ro weighting?                     uint8 {qos}?
        +--ro dependency?                    sn:subscription-id {qos}?
        +--ro (notification-message-origin)?
        |  +--:(interface-originated)
        |  |  +--ro source-interface?              if:interface-ref
        |  +--:(address-originated)
        |     +--ro source-vrf?                    ->
        |  /ni:network-instances/network-instance/name {supports-vrf}?
        |     +--ro source-address?            inet:ip-address-no-zone
        +--ro receivers
           +--ro receiver* [address port]
              +--ro address                   inet:host
              +--ro port                      inet:port-number
              +--ro pushed-notifications?     yang:counter64
              +--ro excluded-notifications?   yang:counter64
              +--ro state                     enumeration
              +---x reset
                 +--ro output
                    +--ro time    yang:date-and-time
```

4. Data Model

```
<CODE BEGINS> file "ietf-subscribed-notifications@2018-02-23.yang"
module ietf-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-interfaces {
    prefix if;
  }
  import ietf-network-instance {
    prefix ni;
  }
  import ietf-restconf   {
    prefix rc;
  }

  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:   Einar Nilsen-Nygaard
               <mailto:einarnn@cisco.com>

     Editor:   Ambika Prasad Tripathy
               <mailto:ambtripa@cisco.com>";

  description
    "Contains a YANG specification for subscribing to event records
    and receiving matching content within notification messages.";

  revision 2018-02-23 {
    description
      "Initial version";
    reference
      "draft-ietf-netconf-subscribed-notifications-10";
  }

  /*
   * FEATURES
   */

  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 configured {
    description
      "This feature indicates that configuration of subscription is
      supported.";
  }

  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 xpath {
    description
      "This feature indicates support for xpath filtering.";
    reference "http://www.w3.org/TR/1999/REC-xpath-19991116";
  }

  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 "draft-ietf-rtgwg-ni-model";
    }

    feature qos {
      description
        "This feature indicates a publisher supports one or more optional
        Quality of Service (QoS) features to differentiate update record
        treatment between publisher and receiver.";
    }

    /*
     * EXTENSIONS
     */

    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.";
    }

    /*
     * IDENTITIES
     */

    /* Identities for RPC and Notification errors */

    identity establish-subscription-error {
       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 delete-subscription-error {
       description
        "Problem found while attempting to fulfill either a
        'delete-subscription' rpc request or a 'kill-subscription'
        rpc request. ";
    }

    identity subscription-terminated-reason {
       description
        "Problem condition communicated to a receiver as part of absolute
        ’subscription-terminated’ notification. ";
    }

    identity subscription-suspended-reason {
       description
        "Problem condition communicated to a receiver as part of absolute
        ’subscription-terminated’ notification. ";
    }

    identity dscp-unavailable {
      base establish-subscription-error;
      description
        "Requested DSCP marking not allocatable.";
    }

    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;
      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.";
    }

    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;
      description
       "Replay cannot be performed for this subscription. This means the
        publisher will not provide the requested historic information from
        the stream via replay to this receiver.";
    }

    identity stream-unavailable {
      base subscription-terminated-reason;
      description
       "Not a subscribable stream.  This means the referenced 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 cannot support the volume of information intended
        to be sent for an existing subscription.";
    }

    /* Identities for encodings */
    identity encodings {
      description
        "Base identity to represent data encodings";
    }

    identity encode-xml {
      base encodings;
      if-feature "encode-xml";
      description
        "Encode data using XML";
    }

    identity encode-json {
      base encodings;
      if-feature "encode-json";
      description
        "Encode data using JSON";
    }

    /* Identities for transports */
    identity transport {
      description
        "An identity that represents a the underlying mechanism for
        passing notification messages.";
    }

    identity netconf {
      base transport;
      description
        "Netconf is used a transport for notification messages and state
         change notifications.";
      reference "draft-ietf-netconf-netconf-event-notifications";
    }

    identity http2 {
      base transport;
      description
        "HTTP2 is used a transport for notification messages and state
         change notifications.";
      reference "draft-ietf-netconf-restconf-notif-03, Sections 3.1.1" +
        "3.1.3";
    }

    identity http1.1 {
      base transport;
      description
        "HTTP1.1 is used a transport for notification messages and state
         change notifications.";
      reference "draft-ietf-netconf-restconf-notif-03, Section 3.1.2";
    }

    /*
     * TYPEDEFs
     */

    typedef subscription-id {
      type uint32;
      description
        "A type for subscription identifiers.";
    }

    typedef filter-id {
      type string;
      description
        "A type to identify filters which can be associated with a
         subscription.";
    }

    typedef encoding {
      type identityref {
        base encodings;
      }
      description
        "Specifies a data encoding, e.g. for a data subscription.";
    }

    typedef transport {
      type identityref {
        base transport;
      }
      description
        "Specifies protocol used to send notification messages to a
         receiver.";
    }

    typedef stream-ref {
      type leafref {
        path "/sn:streams/sn:stream/sn:name";
      }
      description
        "This type is used to reference a system-provided datastream.";
    }

    typedef stream-filter-ref {
      type leafref {
        path "/sn:filters/sn:stream-filter/sn:identifier";
      }
      description
        "This type is used to reference a configured stream filter.";
    }

    /*
     * 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 it 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 it 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 '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 "qos";
        type inet:dscp;
        default "0";
        description
          "The push update's IP packet transport priority. This is made
           visible across network hops to receiver. The 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 priority 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.";
        reference
          "RFC-7540, section 5.3.1";
      }
    }

    grouping subscription-policy-modifiable {
      description
        "This grouping describes all objects which may be changed
        in a subscription via an RPC.";
      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 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 must be for a future time.";
      }
    }

    grouping subscription-policy-dynamic {
      description
        "This grouping describes information concerning a subscription
        which can just be passed over the RPCs defined in this model.";
      leaf encoding {
        type encoding;
        mandatory true;
        description
          "The type of encoding for the subscribed data.";
      }
      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 stream of event records to be considered for
              this subscription.";
          }
          leaf replay-start-time {
            if-feature "replay";
            type yang:date-and-time;
            description
              "Used to trigger the replay feature and indicate that the
              replay should 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, except for
        configured receivers.";
      leaf protocol {
        if-feature "configured";
        type transport;
        mandatory true;
        description
          "This leaf specifies the transport protocol used to deliver
          messages destined to all receivers of a subscription.";
      }
      uses subscription-policy-dynamic;
    }

    grouping notification-origin-info {
      description
        "Defines the sender source from which notification messages for a
         configured subscription are sent.";
      choice notification-message-origin {
        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 {
            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 specfic 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.";
          }
        }
      }
    }

    grouping receiver-info {
      description
        "Defines where and how to get notification messages for a
        configured subscriptions to one or more targeted recipient.  This
        includes specifying the destination addressing as well as a
        transport protocol acceptable to the receiver.";
      container receivers {
        description
          "Set of receivers in a subscription.";
        list receiver {
          key "address port";
          min-elements 1;
          description
            "A single host or multipoint address intended as a target
             for the notification messages of a subscription.";
          leaf address {
            type inet:host;
            description
              "Specifies the address for the traffic to reach a remote
              host. One of the following must be specified: an ipv4
              address, an ipv6 address, or a host name.";
          }
          leaf port {
            type inet:port-number;
            description
              "This leaf specifies the port number to use for messages
               destined for a receiver.";
          }
        }
      }
    }


    /*
     * 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, and 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 {
          refine "encoding" {
            mandatory false;
            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.";
          }
        }
      }
    }


    rc:yang-data establish-subscription-error-stream {
      container establish-subscription-error-stream {
        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.";
        }
        leaf replay-start-time-hint {
          type yang:date-and-time;
            description
              "If a replay has been requested, but the requested replay
              time cannot be honored, this may provide a hint at an
              alternate time which may be supportable.";
        }
      }
    }

    rpc modify-subscription {
      description
        "This RPC allows a subscriber to modify a subscription that was
         previously created using establish-subscription.  If successful,
         the changed subscription remains in effect for the duration of
         the subscriber’s association with the publisher, or until the
         subscription is again modified or 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;
          description
            "Identifier to use for this subscription.";
        }
        uses subscription-policy-modifiable;
      }
    }

    rc:yang-data modify-subscription-error-stream {
      container modify-subscription-error-stream {
        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
          provides 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.";
      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.";
        }
      }
    }

    rpc kill-subscription {
      description
        "This RPC allows an operator to delete a dynamic subscription
        without restrictions on the originating subscriber or underlying
        transport session.";
      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 {
      container delete-subscription-error {
        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.";
        }
      }
    }

    /*
     * NOTIFICATIONS
     */

    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;
      description
        "This notification is sent to indicate that a subscription has
         finished passing event records.";
      leaf identifier {
        type subscription-id;
        mandatory true;
        description
          "This references the gracefully completed subscription.";
      }
    }

    notification subscription-started {
      sn:subscription-state-notification;
      if-feature "configured";
      description
        "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, replay-start-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.";
        }
      }
    }

    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-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-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 .";
    }
  }

  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.";
    }
  }

  /*
   * DATA NODES
   */

  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 streams that are supported by the
         publisher.";
      leaf name {
        type string;
        description
          "A handle for a system-provided datastream 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
           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 {
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp of the creation of the log used to support the
          replay function on this stream. Note that this might be
          earlier then the earliest available information contained in
          the log. This object is updated if the log resets for some
          reason. This object MUST be present if replay is supported.";
      }
      leaf replay-log-aged-time {
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp of the last event record aged out of the log.
          This object MUST be present if replay is supported and any
          event record 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-positioned 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.";
        leaf identifier {
          type subscription-id;
          description
            "Identifier of a subscription; unique within a publisher";
        }
        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.";
        }
        leaf purpose {
          if-feature "configured";
          type string;
          description
            "Open text allowing a configuring entity to embed the
            originator or other specifics of this subscription.";
        }
        uses subscription-policy {
          refine "target/stream/stream" {
            description
              "Indicates the stream of event records to be considered for
              this subscription.   If a 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.";
          }
        }
        uses notification-origin-info {
          if-feature "configured";
        }
        uses receiver-info {
          augment receivers/receiver {
            description
              "include operational data for receivers.";
            leaf pushed-notifications {
              type yang:counter64;
              config false;
              description
                "Operational data which provides the number of update
                 notification messages pushed to a receiver.";
              }
            leaf excluded-notifications {
              type yang:counter64;
              config false;
              description
                "Operational data which provides the number of event
                 records from a stream explicitly removed via filtering so
                 that they are not sent to a receiver.";
              }
            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 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.";
                }
                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 {
              description
                "Allows the reset of this configured subscription receiver
                to the ’connecting’ state. This enables the
                connection process to be reinitiated.";
              output {
                leaf time {
                  type yang:date-and-time;
                  mandatory true;
                  description
                    "Time a publisher returned the receiver to a
                    connecting state.";
                }
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

5. Considerations

5.1. Implementation Considerations

For a deployment including both configured and dynamic subscriptions, split subscription identifiers into static and dynamic halves. That way it is unlikely there will be collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated. The lower half the "identifier" object in the subscriptions container SHOULD be used when the "identifier" is selected and assigned by an external entity (such as with a configured subscription). And the upper half SHOULD be used for subscription identifiers dynamically chosen and assigned by the publisher

Neither state change notification nor subscribed event records within notification messages may be sent before the transport layer, including any required capabilities exchange, has been established.

An implementation may choose to transition between active and suspended subscription states more frequently than required by this specification. However if a subscription is unable to marshal all intended updates into a transmittable message in multiple successive intervals, the subscription 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 an operator wants to let the receivers correlate results, it is useful to use the subscription identifier handle across the receivers to allow that correlation.

5.2. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications
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
Namespace: urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications
Prefix: sn
Reference: draft-ietf-netconf-ietf-subscribed-notifications-08.txt
(RFC form)

5.3. Security Considerations

For dynamic subscriptions the publisher MUST authenticate and authorize all RPC requests.

Subscriptions could overload a publisher’s CPU. For this reason, the publisher MUST have the ability to decline a dynamic subscription request, and provide the appropriate RPC error response to a subscriber should the proposed subscription overly deplete the publisher’s resources.

A publisher needs to be able to suspend an existing dynamic or configured subscription based on capacity constraints. When this occurs, the subscription state MUST be updated accordingly and the receivers notified with subscription state notifications.

If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an excessive use of system resources.

In such a situation, subscription interactions MAY be terminated by terminating the transport session.

For both configured and dynamic subscriptions the publisher MUST authenticate and authorize a receiver via some transport level mechanism before sending any updates.

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 publisher MUST NOT include any content in a notification message for which the receiver has not been authorized.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. No notification messages SHOULD be sent to any receiver which doesn’t even support subscriptions. Receivers that do not want notification messages need only terminate or refuse any transport sessions from the publisher.

The NETCONF Authorization Control Model [RFC6536bis] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-receiver permissions to receive event records from specific streams.

Where NACM is available, the NACM "very-secure" tag MUST be placed on the "kill-subscription" RPC so that only administrators have access to use this.

One subscription id 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 SHOULD NOT be assumed that each receiver is getting identical updates.

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]
Berger, L., Hopps, C., and A. Lindem, "YANG Network Instances", draft-ietf-rtgwg-ni-model-06 (work in progress), January 2018.

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

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

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

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

[RFC6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-09 (work in progress), December 2017.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

7.2. Informative References

[I-D.draft-ietf-netconf-netconf-event-notifications]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", October 2017, <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", January 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", December 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[I.D.draft-ietf-netconf-notification-messages]
Voit, Eric., Clemm, Alexander., Bierman, A., and T. Jenkins, "YANG Notification Headers and Bundles", September 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-notification-messages>.

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

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

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 protocol for all receivers of configured subscriptions.

- QoS section moved in from yang-push

v06 - v07

- Clarification on state machine for configured subscriptions.

v05 - v06

- 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.

- JSON and XML become features. Also Xpath and subtree filtering become features

- Terminology updates with event records, and refinement of filters to just stream filters.

- Encoding refined in establish-subscription so it takes the RPC's encoding as the default.

- 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

- o Subscription status moved under receiver.

v00 - v01

- o Security considerations updated
- o Intro rewrite, as well as scattered text changes
- o Added Appendix A, to help match this to related drafts in progress
- o Updated filtering definitions, and filter types in yang file, and moved to identities for filter types
- o Added Syslog as a stream
- o 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

v00 - v01 of 5277bis

- 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
VMWare

Email: agonzalezpri@vmware.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF A. Clemm
Internet-Draft Huawei
Intended status: Standards Track E. Voit
Expires: August 27, 2018 Cisco Systems
A. Gonzalez Prieto
VMware
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
B. Lengyel
Ericsson
February 23, 2018

YANG Datastore Subscription
draft-ietf-netconf-yang-push-15

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

Copyright Notice

Table of Contents

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

The terms below supplement those defined in [I-D.draft-ietf-netconf-subscribed-notifications]. In addition, the term "datastore" is defined in [I-D.draft-ietf-netmod-revised-datastores].

Datastore node: An instance of management information in a datastore. Also known as "object".

Datastore node update: A data item containing the current value/ property of a datastore node at the time the datastore node update was created.

Datastore subtree: An instantiated datastore node and the datastore nodes that are hierarchically contained within it.

Update record: A representation datastore node update(s) resulting from the application of a selection filter for a subscription. An update record will include the value/property of one or more datastore nodes at a point in time. It may contain the update type for each datastore node (e.g., add, change, delete). Also included may be metadata/headers such as a subscription identifier.

Selection filter: Evaluation and/or selection criteria, which may be applied against a targeted set of objects.

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 for a push update subscription service. This solution supports dynamic as well as configured subscriptions to information updates from datastores. 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 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 triggers for subscriptions: periodic and on-change.

    - For periodic subscriptions, the 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, a 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 of 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.

- An encoding (using anydata) for the contents of periodic and on-change push 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. Promise-Theory 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. If for some reason the publisher of a subscription is not able to keep its promise, receivers MUST be notified immediately and reliably. The publisher MAY also suspend the subscription.

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.

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:

o 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 data 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 [RFC7223]). 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>
 <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <subscription-id>1011</subscription-id>
   <datastore-contents>
     <interfaces-state xmlns="http://foo.com/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>
 <push-change-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <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 ’99999’. 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. Included in these enhancements are error codes which can indicate why a datastore subscription attempt has failed.

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.

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], the NETCONF RPC reply MUST include an "rpc-error" element with the following additional 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" (for error responses to an establish-subscription request), "modify-subscription-error" (for error responses to a modify-subscription request), "delete-subscription-error" (for error responses to a delete-subscription request), "resynch-subscription-error" (for error responses to resynch-subscription request), or "kill-subscription-error" (for error responses to a kill-subscription request), respectively.

- In case of error responses to an establish-subscription or modify-subscription request: optionally, "error-info" containing XML-encoded data with hints regarding parameter settings that might lead to successful requests in the future, per yang-data definitions "establish-subscription-error-datastore" (for error responses to an establish-subscription request) or "modify-subscription-error-datastore (for error responses to a modify-subscription request), respectively. In case of an rpc error as a result of a delete-subscription, or 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 RPC input parameters need to be provided.

For instance, for the following request:

```
<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"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <yp:datastore xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      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:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </establish-subscription>
</netconf:rpc>
```

Figure 3: Establish-Subscription example

the publisher might return:

```
 <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-app-tag>period-unsupported</error-app-tag>
    <error-info>
      <establish-subscription-error-datastore
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
        <yp:period-hint>
          2000
        </yp:period-hint>
      </establish-subscription-error-datastore>
    </error-info>
   </rpc-error>
 </rpc-reply>
```

Figure 4: Error response example

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 lifecyle 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 [RFC6536bis], Section 3.2.4.

```
                      +-----------------+      +--------------------+
  push-update or -->  | datastore node  |  yes | add datastore node |
 push-change-update   | access allowed? | ---> | to update message  |
                      +-----------------+      +--------------------+
```

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., [RFC7223]’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 MAY 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 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. Following YANG tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses with a name in the middle enclose choice and case nodes. New schema objects defined here (i.e., beyond those from [I-D.draft-ietf-netconf-subscribed-notifications]) are identified with "yp".

```
module: ietf-subscribed-notifications
    +--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}?
    +--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}?
    |  +--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 identifier                     subscription-id
          +--ro configured-subscription-state? enumeration {configured}?
          +--rw purpose?                       string {configured}?
          +--rw protocol                       transport {configured}?
          +--rw encoding                       encoding
          +--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}?
          |  |  |        +--:(stream-xpath-filter)
          |  |  |           +--rw stream-xpath-filter?
          |  |  |                                 yang:xpath1.0 {xpath}?
          |  |  +--rw stream                      stream-ref
          |  |  +--rw replay-start-time?    yang:date-and-time {replay}?
          |  +--:(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 stop-time?                       yang:date-and-time
          +--rw dscp?                            inet:dscp {qos}?
          +--rw weighting?                       uint8 {qos}?
          +--rw dependency?                      subscription-id {qos}?
          +--rw (notification-message-origin)?
          |  +--:(interface-originated)
          |  |  +--rw source-interface?                if:interface-ref
          |  +--:(address-originated)
          |     +--rw source-vrf?                      ->
          |  /ni:network-instances/network-instance/name {supports-vrf}?
          |     +--rw source-address?            inet:ip-address-no-zone
          +--rw receivers
          |  +--rw receiver* [address port]
          |     +--rw address                   inet:host
          |     +--rw port                      inet:port-number
          |     +--ro pushed-notifications?     yang:counter64
          |     +--ro excluded-notifications?   yang:counter64
          |     +--ro status                    enumeration
          |     +---x reset
          |        +--ro output
          |           +--ro time    yang:date-and-time
          +--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 encoding?                      encoding
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |  |  +---w (stream-filter)?
    |  |  |  |  |  +--:(by-reference)
    |  |  |  |  |  |  +---w stream-filter-ref          stream-filter-ref
    |  |  |  |  |  +--:(within-subscription)
    |  |  |  |  |     +---w (filter-spec)?
    |  |  |  |  |        +--:(stream-subtree-filter)
    |  |  |  |  |        |  +---w stream-subtree-filter?
    |  |  |  |  |                                   <anydata> {subtree}?
    |  |  |  |  |        +--:(stream-xpath-filter)
    |  |  |  |  |           +---w stream-xpath-filter?
    |  |  |  |  |                                 yang:xpath1.0 {xpath}?
    |  |  |  |  +---w stream                      stream-ref
    |  |  |  |  +---w replay-start-time?    yang:date-and-time {replay}?
    |  |  |  +--:(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}?
    |  |  |              +--:(yp:datastore-xpath-filter)
    |  |  |                 +---w yp:datastore-xpath-filter?
    |  |  |                                    yang:xpath1.0 {sn:xpath}?
    |  |  +---w stop-time?                     yang:date-and-time
    |  |  +---w dscp?                          inet:dscp {qos}?
    |  |  +---w weighting?                     uint8 {qos}?
    |  |  +---w dependency?                    subscription-id {qos}?
    |  |  +---w (yp:update-trigger)?
    |  |     +--:(yp:periodic)
    |  |     |  +---w yp:periodic!
    |  |     |     +---w yp:period         yang:timeticks
    |  |     |     +---w yp:anchor-time?   yang:date-and-time
    |  |     +--:(yp:on-change) {on-change}?
    |  |        +---w yp:on-change!
    |  |           +---w yp:dampening-period?    yang:timeticks
    |  |           +---w yp:no-synch-on-start?   empty
    |  |           +---w yp:excluded-change*     change-type
    |  +--ro output
    |     +--ro identifier    subscription-id
    +---x modify-subscription
    |  +---w input
    |     +---w identifier?                    subscription-id
    |     +---w (target)
    |     |  +--:(stream)
    |     |  |  +---w (stream-filter)?
    |     |  |     +--:(by-reference)
    |     |  |     |  +---w stream-filter-ref          stream-filter-ref
    |     |  |     +--:(within-subscription)
    |     |  |        +---w (filter-spec)?
    |     |  |           +--:(stream-subtree-filter)
    |     |  |           |  +---w stream-subtree-filter?
    |     |  |                                      <anydata> {subtree}?
    |     |  |           +--:(stream-xpath-filter)
    |     |  |              +---w stream-xpath-filter?
    |     |  |                                    yang:xpath1.0 {xpath}?
    |     |  +--:(yp:datastore)
    |     |     +---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}?
    |     |              +--:(yp:datastore-xpath-filter)
    |     |                 +---w yp:datastore-xpath-filter?
    |     |                                    yang:xpath1.0 {sn:xpath}?
    |     +---w stop-time?                     yang:date-and-time
    |     +---w (yp:update-trigger)?
    |        +--:(yp:periodic)
    |        |  +---w yp:periodic!
    |        |     +---w yp:period         yang:timeticks
    |        |     +---w yp:anchor-time?   yang:date-and-time
    |        +--:(yp:on-change) {on-change}?
    |           +---w yp:on-change!
    |              +---w yp:dampening-period?   yang:timeticks
    +---x delete-subscription
    |  +---w input
    |     +---w identifier    subscription-id
    +---x kill-subscription
       +---w input
          +---w identifier    subscription-id

  yang-data (for placement into rpc error responses)
    +-- establish-subscription-error-stream
    |  +--ro reason?                   identityref
    |  +--ro filter-failure-hint?      string
    |  +--ro replay-start-time-hint?   yang:date-and-time
    +-- modify-subscription-error-stream
       +--ro reason?                identityref
       +--ro filter-failure-hint?   string

  notifications:
    +---n replay-completed {replay}?
    |  +--ro identifier    subscription-id
    +---n subscription-completed
    |  +--ro identifier    subscription-id
    +---n subscription-started {configured}?
    |  +--ro identifier                     subscription-id
    |  +--ro protocol                       transport {configured}?
    |  +--ro encoding                       encoding
    |  +--ro (target)
    |  |  +--:(stream)
    |  |  |  +--ro (stream-filter)?
    |  |  |  |  +--:(by-reference)
    |  |  |  |  |  +--ro stream-filter-ref             stream-filter-ref
    |  |  |  |  +--:(within-subscription)
    |  |  |  |     +--ro (filter-spec)?
    |  |  |  |        +--:(stream-subtree-filter)
    |  |  |  |        |  +--ro stream-subtree-filter?
    |  |  |  |        |                            <anydata> {subtree}?
    |  |  |  |        +--:(stream-xpath-filter)
    |  |  |  |           +--ro stream-xpath-filter?
    |  |  |  |                                    yang:xpath1.0 {xpath}?
    |  |  |  +--ro stream                         stream-ref
    |  |  |  +--ro replay-start-time?       yang:date-and-time {replay}?
    |  |  +--:(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}?
    |  |              +--:(yp:datastore-xpath-filter)
    |  |                 +--ro yp:datastore-xpath-filter?
    |  |                                    yang:xpath1.0 {sn:xpath}?
    |  +--ro stop-time?                     yang:date-and-time
    |  +--ro dscp?                          inet:dscp {qos}?
    |  +--ro weighting?                     uint8 {qos}?
    |  +--ro dependency?                    subscription-id {qos}?
    |  +--ro (yp:update-trigger)?
    |     +--:(yp:periodic)
    |     |  +--ro 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
    |  +--ro identifier    subscription-id
    +---n subscription-modified
    |  +--ro identifier                     subscription-id
    |  +--ro protocol                       transport {configured}?
    |  +--ro encoding                       encoding
    |  +--ro (target)
    |  |  +--:(stream)
    |  |  |  +--ro (stream-filter)?
    |  |  |  |  +--:(by-reference)
    |  |  |  |  |  +--ro stream-filter-ref             stream-filter-ref
    |  |  |  |  +--:(within-subscription)
    |  |  |  |     +--ro (filter-spec)?
    |  |  |  |        +--:(stream-subtree-filter)
    |  |  |  |        |  +--ro stream-subtree-filter?
    |  |  |  |        |                             <anydata> {subtree}?
    |  |  |  |        +--:(stream-xpath-filter)
    |  |  |  |           +--ro stream-xpath-filter?
    |  |  |  |                                    yang:xpath1.0 {xpath}?
    |  |  |  +--ro stream                         stream-ref
    |  |  |  +--ro replay-start-time?       yang:date-and-time {replay}?
    |  |  +--:(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}?
    |  |              +--:(yp:datastore-xpath-filter)
    |  |                 +--ro yp:datastore-xpath-filter?
    |  |                                    yang:xpath1.0 {sn:xpath}?
    |  +--ro stop-time?                     yang:date-and-time
    |  +--ro dscp?                          inet:dscp {qos}?
    |  +--ro weighting?                     uint8 {qos}?
    |  +--ro dependency?                    subscription-id {qos}?
    |  +--ro (yp:update-trigger)?
    |     +--:(yp:periodic)
    |     |  +--ro 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-terminated
    |  +--ro identifier    subscription-id
    |  +--ro reason        identityref
    +---n subscription-suspended
       +--ro identifier    subscription-id
       +--ro reason        identityref

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
    +-- modify-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 and enhanced 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 [I-D.draft-ietf-netmod-revised-datastores]. 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 many be calculated from the periodic parameters:

  * a "period" which defines duration between period 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]. Some 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"

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" object. This object 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:

```
<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"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore>
      <yp:source xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        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

The publisher MUST respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the "identifier" of the accepted subscription. In that case a publisher MAY respond:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
       ok
    </subscription-result>
    <identifier
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
       52
    </identifier>
</rpc-reply>
```

Figure 8: Establish-subscription positive RPC response

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:

```
<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"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      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:

```
<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-message>
    <error-path
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      /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"
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <identifier>1011</identifier>
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      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:

o "error-type" of "application".

o "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 "modify-subscription-error".

- "error-path" pointing to the object or parameter that caused the rejection.

- 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:

```
<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"
  xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <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@2018-02-23.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;
  }
  import ietf-subscribed-notifications {
    prefix sn;
  }
  import ietf-datastores {
    prefix ds;
  }
  import ietf-restconf   {
    prefix rc;
  }

  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:einarnn@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 2018-02-23 {
    description
      "Initial revision.";
    reference
      "draft-ietf-netconf-yang-push-15";
  }

 /*
  * FEATURES
  */

  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 when this expressed.";
  }

  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 when this expressed.";
  }

  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.

      Where this identity is referenced as an 'error-app-tag' within an
      RPC response, the response's 'error-info' may contain:";
  }
  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.";
  }

  /*
   * TYPE DEFINITIONS
   */

  typedef change-type {
    type enumeration {
      enum "create" {
        description
          "Create a new data resource if it does not already exist.  If
          it already exists, replace it.";
      }
      enum "delete" {
        description
          "Delete a data resource if it already exists. If it does not
          exist, take no action.";
      }
      enum "insert" {
        description
          "Insert a new user-ordered data resource";
      }
      enum "merge" {
        description
          "merge the edit value with the target data resource; create
          if it does not already exist";
      }
      enum "move" {
        description
          "Reorder the target data resource";
      }
      enum "replace" {
        description
          "Replace the target data resource with the edit value";
      }
      enum "remove" {
        description
          "Remove a data resource if it already exists ";
      }
    }
    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:identifier";
    }
    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:

           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.";
      }
    }
  }

  grouping selection-filter-objects {
    description
      "This grouping defines a selector for objects from a
       datastore.";
    choice selection-filter {
      description
        "The source of the selection filter applied to the subscription.
        This will come either referenced from a global list, or be
        provided within the subscription itself.";
      case by-reference {
        description
          "Incorporate a filter that has been configured separately.";
        leaf selection-filter-ref {
          type selection-filter-ref;
          mandatory true;
          description
            "References an existing selection filter which is to be
             applied to the subscription.";
        }
      }
      case within-subscription {
        description
          "Local definition allows a filter to have the same lifecycle
          as the subscription.";
        uses selection-filter-types;

      }
    }
  }

  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
           publisher’s ability to service for this subscription.";
      }
    }


    /*
     * RPCs
     */

     rpc resynch-subscription {
      if-feature "on-change";
      description
        "This RPC allows a subscriber of an active on-change
         subscription to request a full push of objects in their current
         state. A successful result would invoke a push-update of all
         datastore objects that the subscriber is permitted to access.
         This request may only come from the same subscriber using the
         establish-subscription RPC.";
      input {
        leaf identifier {
          type sn:subscription-id;
          mandatory true;
          description
            "Identifier of the subscription that is to be resynched.";
        }
      }
    }

    rc: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-error-datastore {
      container establish-subscription-error-datastore {
        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-error-datastore {
      container modify-subscription-error-datastore {
        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.";
      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 format and syntax of the data
           corresponds to the format and syntax of data 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 needed
           to update a remote datastore starting at the time of the
           previous update, per the terms of the subscription.  Changes
           are encoded analogous to the syntax of a corresponding yang-
           patch operation, i.e. a yang-patch operation applied to the
           datastore implied by the previous update to result in the
           current state.";
        reference
          "RFC 8072 section 2.5, with a delta that it is ok to receive
           ability create on an existing node, or receive a delete on a
           missing node.";
      }
    }

    augment "/sn:subscription-started" {
      description
        "This augmentation adds many 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 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:subscription-modified" {
      description
        "This augmentation adds many 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.";
          }
        }
      }
    }

    /*
     * DATA NODES
     */

    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 "identifier";
        description
          "A list of pre-positioned filters that can be applied
           to datastore subscriptions.";
        leaf identifier {
          type sn:filter-id;
          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;
      }
    }
  }

  <CODE ENDS>
```

6. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-push
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
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-push
Prefix: yp
Reference: draft-ietf-netconf-yang-push-12.txt (RFC form)

7. Security Considerations

All security considerations from [I-D.draft-ietf-netconf-subscribed-notifications] are relevant for datastores. In addition there are specific security considerations for receivers defined in Section 3.9

If the access control permissions on subscribed YANG nodes change during the lifecycle of a subscription, a publisher MUST either transparently conform to the new access control permissions, or must terminate or restart the subscriptions so that new access control permissions are re-established.

The NETCONF Authorization Control Model SHOULD be used to restrict the delivery of YANG nodes for which the receiver has no access.

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Martin Bjorklund, and Guangying Zheng.

9. References

9.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-06 (work in progress), January 2018.

[I-D.draft-ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-04 (work in progress), August 2017.

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

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

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

[RFC6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-05 (work in progress), September 2017.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/info/rfc8072>.

9.2. Informative References

[I-D.draft-ietf-netconf-netconf-event-notifications] Gonzalez Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF Support for Event Notifications", September 2017.

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

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

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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):

```
establish-subscription         modify-subscription
----------------------         -------------------
 cant-exclude                   sn:filter-unsupported
 datastore-not-subscribable     sn:insufficient-resources
 sn:dscp-unavailable            sn:no-such-subscription
 sn:filter-unsupported          period-unsupported
 sn:insufficient-resources      result-too-big
 on-change-unsupported          synchronization-size
 on-change-synch-unsupported    unchanging-selection
 period-unsupported
 result-too-big                resynch-subscription
 synchronization-size          --------------------
 unchanging-selection           no-such-subscription-resynch
                                synchronization-size

delete-subscription            kill-subscription
----------------------         -----------------
 sn:no-such-subscription        sn:no-such-subscription
```

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 "modifiy-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)

v14 - v15

- 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

- Minor text fixes.

v12 - v13

- Hint negotiation models now show error examples.

- yang-data structures for rpc errors.

v11 - v12

- Included Martin's review clarifications.

- QoS moved to subscribed-notifications

- time-of-update removed as it is redundant with RFC5277's eventTime, and other times from notification-messages.

- Error model moved to match existing implementations

- On-change notifiable removed, how to do this is implementation specific.

- NMDA model supported. Non NMDA version at https://github.com/netconf-wg/yang-push/

v10 - v11

- 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:datastore 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 descibing 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

- 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

- Client and Server changed to subscriber, publisher, and receiver

- Anchor time for periodic

- Message format for synchronization (i.e. synch-on-start)

- Material moved into 5277bis

- QoS parameters supported, by not allowed to be modified by RPC

- 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-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com

NETCONF Working Group   K. Watsen
Internet-Draft   Juniper Networks
Intended status: Standards Track   March 5, 2018
Expires: September 6, 2018

Common YANG Data Types for Cryptography
draft-kwatsen-netconf-crypto-types-00

Abstract

This document defines a YANG identities, typedefs, and groupings useful for when working with ASN.1 structures, algorithms, and private keys.

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:

- "XXXX" --> 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:

- "2018-03-05" --> 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

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG identities, typedefs, and groupings useful for when working with ASN.1 structures, algorithms, and private keys.

1.1. 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.

1.2. Tree Diagram Notation

Tree diagrams used in this document follow the notation defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Tree Diagram

The following tree diagram provides an overview of the groupings, actions, and notifications defined in the ietf-crypto-types YANG module. The YANG module defines many identities and typedefs that are not represented by this tree diagram.

```
   module: ietf-crypto-types

     notifications:
       +---n certificate-expiration
          +--ro certificate        instance-identifier
          +--ro expiration-date    yang:date-and-time

     grouping certificates-grouping
       +---- certificates
       |  +---- certificate* [name]
       |     +---- name?    string
       |     +---- value?   binary
       +---x generate-certificate-signing-request
          +---w input
          |  +---w subject       binary
          |  +---w attributes?   binary
          +--ro output
             +--ro certificate-signing-request    binary
     grouping private-key-grouping
       +---- algorithm?             identityref
       +---- private-key?           union
       +---- public-key?            binary
       +---x generate-private-key
          +---w input
             +---w algorithm    identityref
```

3. Examples

These examples illustrate the use of the groupings, actions, and notifications defined in the ietf-crypto-types YANG module. The YANG module defines many identities and typedefs that are not represented that are not represented by these examples.

3.1. Private Key and Associated Certificate Configuration

The following example illustrates a configured private key along with an associated certificate. This example uses the "ex-crypto-types-usage" module defined in Appendix A.

```
  [ note: '\' line wrapping for formatting only]

  <key xmlns="http://example.com/ns/example-crypto-types-usage">
    <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>
    <certificates>
      <certificate>
        <name>domain certificate</name>
        <value>base64encodedvalue==</value>
      </certificate>
    </certificates>
   </key>
```

3.2. Certificate Signing Request Action

The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol. This example uses the "ex-crypto-types-usage" module defined in Appendix A.

```
  REQUEST
  -------
  <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <action xmlns="urn:ietf:params:xml:ns:yang:1">
      <key xmlns="http://example.com/ns/example-crypto-types-usage">
        <generate-certificate-signing-request>
          <subject>base64encodedvalue==</subject>
          <attributes>base64encodedvalue==</attributes>
        </generate-certificate-signing-request>
      </key>
    </action>
  </rpc>

  RESPONSE
  --------
  <rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <certificate-signing-request
       xmlns="http://example.com/ns/example-crypto-types-usage">
       base64encodedvalue==
     </certificate-signing-request>
  </rpc-reply>
```

3.3. Generate Private Key Action

The following example illustrates the "generate-private-key" action in use with the NETCONF protocol. This example uses the "ex-crypto-types-usage" module defined in Appendix A.

```
REQUEST
-------
[ note: '\' line wrapping for formatting only]

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0\
">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <key xmlns="http://example.com/ns/example-crypto-types-usage">
      <generate-private-key>
        <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto\
-types">ct:secp521r1</algorithm>
      </generate-private-key>
    </key>
  </action>
</rpc>

RESPONSE
--------
<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

3.4. Certificate Expiration Notification

The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol. This example uses the "ex-crypto-types-usage" module defined in Appendix A.

[note: '\' line wrapping for formatting only]

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-07-08T00:01:00Z</eventTime>
  <certificate-expiration
    xmlns="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
    <certificate xmlns:ex="http://example.com/ns/example-crypto-type\
s-usage">
      /ex:key/ex:certificates/ex:certificate[ex:name='domain certifi\
cate']
    </certificate>
    <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
  </certificate-expiration>
</notification>
```

4. YANG Module

This YANG module imports modules defined in [RFC6536] and [RFC6991]. This module uses data types defined in [RFC2315], [RFC2986], [RFC3447], [RFC4253], [RFC5280], [RFC5915], and [ITU.X690.1994]. This module uses algorithms defined in [RFC3447] and [RFC5480].

```
<CODE BEGINS> file "ietf-crypto-types@2018-03-05.yang"
module ietf-crypto-types {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-crypto-types";
  prefix "ct";

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

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 6536: Network Configuration Protocol (NETCONF) Access
       Control Model";
  }

  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 common YANG types for cryptography applications.

      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 "2018-03-05" {
      description
       "Initial version";
      reference
       "RFC XXXX: Common YANG Data Types for Cryptography";
    }


    /*****************************************/
    /*   Identities for Hashing Algorithms  */
    /*****************************************/

    identity hash-algorithm {
      description
        "A base identity for hash algorithm verification.

         This identity is used in the ietf-zerotouch-information
         module (draft-ietf-netconf-zerotouch)";
    }

    identity sha-256 {
      base "hash-algorithm";
      description "The SHA-256 algorithm.";
      reference "RFC 6234: US Secure Hash Algorithms.";
    }
```

```
  /************************************************************/
  /*   Identities for Key Algorithms (used by Certificates)   */
  /************************************************************/

  identity key-algorithm {
    description
      "Base identity from which all key-algorithms are derived.

       This identity is used in the 'private-key-grouping' grouping
       and the 'generate-private-key' action below.";
  }

  identity rsa1024 {
    base key-algorithm;
    description
      "The RSA algorithm using a 1024-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa2048 {
    base key-algorithm;
    description
      "The RSA algorithm using a 2048-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa3072 {
    base key-algorithm;
    description
      "The RSA algorithm using a 3072-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa4096 {
    base key-algorithm;
    description
      "The RSA algorithm using a 4096-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }
  identity rsa7680 {
    base key-algorithm;
    description
      "The RSA algorithm using a 7680-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa15360 {
    base key-algorithm;
    description
      "The RSA algorithm using a 15360-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity secp192r1 {
    base key-algorithm;
    description
      "The secp192r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp256r1 {
    base key-algorithm;
    description
      "The secp256r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp384r1 {
    base key-algorithm;
    description
      "The secp384r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp521r1 {
    base key-algorithm;
    description
```

```
      "The secp521r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }


  /*************************************/
  /*   Typedefs for ASN.1 structures  */
  /*************************************/

  typedef x509 {
    type binary;
    description
      "A Certificate structure, as specified in RFC 5280, 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).";
  }


  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).";
  }



  /**************************************************************/
  /*   Groupings for a Private Key and its Associated Certificates  */
  /**************************************************************/

  grouping private-key-grouping {
    description
      "A private/public key pair, and an action to request the
       system to generate a private key.

       This grouping is currently used by the YANG modules
       ietf-ssh-client, ietf-ssh-server, ietf-tls-client,
       and ietf-tls-server, where it populates the SSH/TLS
       peer object’s private key parameters.";
    leaf algorithm {
      type identityref {
        base "key-algorithm";
      }
      description
        "Identifies the key’s algorithm.  More specifically, this
         leaf specifies how the ’private-key’ and ’public-key’
         binary leafs are encoded.";
    }
    leaf private-key {
      nacm:default-deny-all;
      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).";
          }
        }
      }
      must "../algorithm";
      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
         [RFC3447], and an Elliptic Curve Cryptography (ECC) key
         is represented as ECPrivateKey as defined in [RFC5915]";
      reference
        "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                   RSA Cryptography Specifications Version 2.1.
         RFC 5915: Elliptic Curve Private Key Structure.";
    }
    leaf public-key {
      type binary;
      must "../algorithm";
      must "../private-key";
      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
         [RFC3447], and an Elliptic Curve Cryptography (ECC) key
         is represented using the 'publicKey' described in
         [RFC5915]";
      reference
        "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                   RSA Cryptography Specifications Version 2.1.
         RFC 5915: Elliptic Curve Private Key Structure.";
    }
    action generate-private-key {
      description
        "Requests the device to generate a private key using the
         specified key algorithm.  This action is primarily to
         support cryptographic processors that must generate
         the private key themselves.  The resulting key is
         considered operational state and hence only present
         in <operational>.";
      input {
        leaf algorithm {
          type identityref {
            base "key-algorithm";
          }
          mandatory true;
          description
            "The algorithm to be used when generating the key.";
        }
      }
    } // end generate-private-key
  }



  grouping certificates-grouping {
    description
      "A container of certificates, and an action to generate
       a certificate signing request.

       This grouping is currently used by the YANG modules
       ietf-ssh-client, ietf-ssh-server, ietf-tls-client,
       and ietf-tls-server, where it populates the SSH/TLS
       peer object's value for a certificates associated
       with the private key.";
    container certificates {
      description
        "Certificates associated with this key.  More than one
         certificate supports, for instance, a TPM-protected
         key that has both IDevID and LDevID certificates
         associated.";
      list certificate {
        key name;
        description
          "A certificate for this private key.";
        leaf name {
          type string;
          description
            "An arbitrary name for the certificate.";
        }
        leaf value {
          type binary;
          description
           "A PKCS #7 SignedData structure, as specified by
            Section 9.1 in RFC 2315, containing just certificates
            (no content, signatures, or CRLs), encoded using ASN.1
            distinguished encoding rules (DER), as specified in
            ITU-T X.690.

            This structure contains the certificate itself as well
            as any intermediate certificates leading up to a trust
            anchor certificate.  The trust anchor certificate MAY
            be included as well.";
          reference
            "RFC 2315:
               PKCS #7: Cryptographic Message Syntax Version 1.5.
             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).";
        }
      }
    }
    action generate-certificate-signing-request {
      description
        "Generates a certificate signing request structure for
         the associated private 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 from 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.";
          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).";
        }
        leaf attributes {
          type binary;
          description
           "The ’attributes’ field from 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.";
          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 4.1 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).";

        }
      }
    }
  }


  notification certificate-expiration {
    description
      "A notification indicating that a 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.";
    leaf certificate {
      type instance-identifier;
      mandatory true;
      description
        "Identifies which certificate is expiring or is expired.";
    }
    leaf expiration-date {
      type yang:date-and-time;
      mandatory true;
      description
        "Identifies the expiration date on the certificate.";
    }
  }

}
<CODE ENDS>
```

5. Security Considerations

TBD

6. IANA Considerations

6.1. 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:

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

6.2. The YANG Module Names Registry

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

```
name:         ietf-crypto-types
namespace:    urn:ietf:params:xml:ns:yang:ietf-crypto-types
prefix:       ct
reference:    RFC XXXX
```

7. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name):

8. References

8.1. Normative References

[ITU.X690.1994]
International Telecommunications 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, 1994.

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

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <https://www.rfc-editor.org/info/rfc3447>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

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

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC5915] Turner, S. and D. Brown, "Elliptic Curve Private Key Structure", RFC 5915, DOI 10.17487/RFC5915, June 2010, <https://www.rfc-editor.org/info/rfc5915>.

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

8.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Example YANG Module

The following example YANG module has been constructed to illustrate the groupings defined in the "ietf-crypto-types" module. This YANG module is uses as a basis for the protocol examples in Section 3.

```
module ex-crypto-types-usage {
  yang-version 1.1;

  namespace "http://example.com/ns/example-crypto-types-usage";
  prefix "ctu";

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

  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 illustrates using groupings defined in the YANG
    module ietf-crypto-types.";

  revision "YYYY-MM-DD" {
    description
     "Initial version";
  }

  container key {
    uses ct:private-key-grouping;
    uses ct:certificates-grouping;
    description
      "A container of certificates, and an action to generate
       a certificate signing request.";
  }
}
```

Appendix B. Change Log

TBD

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

K. Watsen
Juniper Networks

March 5, 2018

YANG Data Model for Global Trust Anchors
draft-kwatsen-netconf-trust-anchors-00

Abstract

This document defines a YANG data model for configuring global sets of X.509 certificates and SSH host-keys that can be referenced by other data models for trust. While the SSH host-keys are uniquely for the SSH protocol, the X.509 certificates may be used for multiple uses, including authenticating protocol peers and verifying signatures.

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:

o "XXXX" --> 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:

o "2018-03-05" --> 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

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG data model for configuring global sets of X.509 certificates and SSH host-keys that can be referenced by other data models for trust. While the SSH host-keys are uniquely for the SSH protocol, the X.509 certificates may be used for multiple uses, including authenticating protocol peers and verifying signatures.

1.1. 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.

1.2. Tree Diagram Notation

Tree diagrams used in this document follow the notation defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Tree Diagram

The following tree diagram provides an overview of the "ietf-trust-anchors" module.

```
module: ietf-trust-anchors
  +--rw trust-anchors
     +--rw pinned-certificates* [name]
     |  +--rw name                  string
     |  +--rw description?          string
     |  +--rw pinned-certificate* [name]
     |     +--rw name    string
     |     +--rw data    binary
     +--rw pinned-host-keys* [name]
        +--rw name               string
        +--rw description?       string
        +--rw pinned-host-key* [name]
           +--rw name    string
           +--rw data    binary
```

3. Example Usage

The following example illustrates configured data.

```
<trust-anchors xmlns="urn:ietf:params:xml:ns:yang:ietf-trust-anchors">

  <!-- Manufacturer’s trust root CA certs -->
  <pinned-certificates>
    <name>manufacturers-root-ca-certs</name>
    <description>
      Certificates built into the device for authenticating
      manufacturer-signed objects, such as TLS server certificates,
      vouchers, etc..  Note, though listed here, these are not
      configurable; any attempt to do so will be denied.
    </description>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 1</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 2</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- pinned netconf/restconf client certificates -->
  <pinned-certificates>
    <name>explicitly-trusted-client-certs</name>
    <description>
      Specific client authentication certificates for explicitly
      trusted clients.  These are needed for client certificates
      that are not signed by a pinned CA.
    </description>
    <pinned-certificate>
      <name>George Jetson</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- pinned netconf/restconf server certificates -->
  <pinned-certificates>
    <name>explicitly-trusted-server-certs</name>
    <description>
      Specific server authentication certificates for explicitly
      trusted servers.  These are needed for server certificates
      that are not signed by a pinned CA.
    </description>
    <pinned-certificate>
      <name>Fred Flintstone</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- trust anchors (CA certs) for authenticating clients -->
  <pinned-certificates>
    <name>deployment-specific-ca-certs</name>
    <description>
      Trust anchors (i.e. CA certs) that are used to authenticate
      client connections.  Clients are authenticated if their
      certificate has a chain of trust to one of these configured
      CA certificates.
    </description>
    <pinned-certificate>
      <name>ca.example.com</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- trust anchors for random HTTPS servers on Internet -->
  <pinned-certificates>
    <name>common-ca-certs</name>
    <description>
      Trusted certificates to authenticate common HTTPS servers.
      These certificates are similar to those that might be
      shipped with a web browser.
    </description>
    <pinned-certificate>
      <name>ex-certificate-authority</name>
      <data>base64encodedvalue==</data>
    </pinned-certificate>
  </pinned-certificates>

  <!-- pinned SSH host keys -->
  <pinned-host-keys>
    <name>explicitly-trusted-ssh-host-keys</name>
    <description>
      Trusted SSH host keys used to authenticate SSH servers.
      These host keys would be analogous to those stored in
      a known_hosts file in OpenSSH.
    </description>
    <pinned-host-key>
      <name>corp-fw1</name>
      <data>base64encodedvalue==</data>
    </pinned-host-key>
  </pinned-host-keys>

</trust-anchors>
```

4. YANG Module

This YANG module imports modules defined in [RFC6536]. This module uses data types defined in [RFC2315], [RFC4253], [RFC5280], and [ITU.X690.1994].

```
   <CODE BEGINS> file "ietf-trust-anchors@2018-03-05.yang"
   module ietf-trust-anchors {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-trust-anchors";
     prefix "ta";

     import ietf-netconf-acm {
       prefix nacm;
       reference
         "RFC 6536: Network Configuration Protocol (NETCONF) Access
          Control Model";
     }

     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 a data model for configuring global
       trust anchors used by other data models.  The data model
       actually enables the configuration of sets of trust
       anchors.  This data model supports configuring trust
       anchors for both X.509 certificates and SSH host keys.

       This data model does not support the configuring trust
       anchors for SSH client keys, or pinning of the client
       keys themselves, as the ability to do so is already
       supported by ietf-system in RFC 7317.


       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 "2018-03-05" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Data Model for Global Trust Anchors";
     }

     // Identities


     // typedefs

     typedef pinned-certificates {
       type leafref {
         path "/ta:trust-anchors/ta:pinned-certificates/ta:name";
       }
       description
         "This typedef enables importing modules to easily define a
          reference to pinned-certificates.  Use of this type also
          impacts the YANG tree diagram output.";
     }

     typedef pinned-host-keys {
       type leafref {
         path "/ta:trust-anchors/ta:pinned-host-keys/ta:name";
       }
       description
         "This typedef enables importing modules to easily define a
          reference to pinned-host-keys.  Use of this type also
          impacts the YANG tree diagram output.";
       reference
         "I-D.ietf-netmod-yang-tree-diagrams: YANG Tree Diagrams";
     }


     // protocol accessible nodes

     container trust-anchors {
       nacm:default-deny-write;
       description
         "Contains sets of X.509 certificates and SSH host keys.";

       list pinned-certificates {
         key name;
         description
           "A list of pinned certificates.  These certificates can be
            used by a server to authenticate clients, or by a client to
            authenticate servers.   Each list of pinned certificates
            SHOULD be specific to a purpose, as the list as a whole
            may be referenced by other modules.  For instance, a
            NETCONF server's configuration might use a specific list
            of pinned certificates for when authenticating NETCONF
            client connections.";
         leaf name {
           type string;
           description
             "An arbitrary name for this list of pinned certificates.";
         }
         leaf description {
           type string;
           description
             "An arbitrary description for this list of pinned
              certificates.";
         }
         list pinned-certificate {
           key name;
           description
             "A pinned certificate.";
           leaf name {
             type string;
             description
               "An arbitrary name for this pinned certificate. The
                name must be unique across all lists of pinned
                certificates (not just this list) so that leafrefs
                from another module can resolve to unique values.";
           }
           leaf data {
             type binary;
             mandatory true;
             description
               "An X.509 v3 certificate structure as specified by RFC
                5280, Section 4 encoded using the ASN.1 distinguished
                encoding rules (DER), as specified in ITU-T X.690.";
             reference
               "RFC 5280:
                  Internet X.509 Public Key Infrastructure Certificate
                  and Certificate Revocation List (CRL) Profile.
                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).";
           }
         }
       }

       list pinned-host-keys {
         key name;
         description
           "A list of pinned host keys.  These pinned host-keys can
            be used by clients to authenticate SSH servers.  Each
            list of pinned host keys SHOULD be specific to a purpose,
            so the list as a whole may be referenced by other modules.
            For instance, a NETCONF client’s configuration might
            point to a specific list of pinned host keys for when
            authenticating specific SSH servers.";
         leaf name {
           type string;
           description
             "An arbitrary name for this list of pinned SSH host keys.";
         }
         leaf description {
           type string;
           description
             "An arbitrary description for this list of pinned SSH host
              keys.";
         }
         list pinned-host-key {
           key name;
           description
             "A pinned host key.";
           leaf name {
             type string;
             description
               "An arbitrary name for this pinned host-key. Must be
                unique across all lists of pinned host-keys (not just
                this list) so that a leafref to it from another module
                can resolve to unique values.";
           }
           leaf data {
             type binary;
             mandatory true;
             description
               "The binary public key data for this SSH key, as
                specified by RFC 4253, Section 6.6, i.e.:

                  string    certificate or public key format
                            identifier
                  byte[n]   key/certificate data.";
             reference
               "RFC 4253: The Secure Shell (SSH) Transport Layer
                          Protocol";
           }
         }
       }
```

```
     }

   }
   <CODE ENDS>
```

5. Security Considerations

TBD

6. IANA Considerations

6.1. 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:

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

6.2. The YANG Module Names Registry

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

```
name:         ietf-trust-anchors
namespace:    urn:ietf:params:xml:ns:yang:ietf-trust-anchors
prefix:       ta
reference:    RFC XXXX
```

7. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name):

8. References

8.1. Normative References

[ITU.X690.1994]
International Telecommunications 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, 1994.

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

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

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

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

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

8.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Change Log

TBD

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

```
Network Working Group                                         B. Lengyel
Internet-Draft                                                  Ericsson
Intended status: Standards Track                                A. Clemm
Expires: August 26, 2018                                      Huawei USA
                                                       February 22, 2018
```

YangPush Notification Capabilities
draft-lengyel-netconf-notification-capabilities-00

Abstract

This document proposes a YANG module that allows a YANG server to specify for which data nodes it will send "YANG Datastore Subscription" on-change notifications. It also proposes to use YANG Instance Data to document this information in implementation time.

Status of This Memo

Copyright Notice

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.

On-change Notification Capability: The capability of the YANG server to send on-change notifications on the change of the value for a specific data node.

Implementation-time information: Information about the YANG server’s behavior that is made available during the implementation of the server, available from a source other then a running Yang server .

Rutime-information: Information about the YANG server’s behavior that is available from the running YANG server via a protocol like NETCONF, RESTCONF or HTTPS.

2. Introduction

As defined in [I-D.ietf-netconf-yang-push] a YANG server may allow clients to subscribe to updates from a datastore and subsequently push such update notifications to the client. Notifications may be sent periodically or on-change (more or less immendiately after each change).

In some cases, a publisher supporting on-change notifications will 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. 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.

Faced with the reality that support for on-change notification does not mean that such notifications will be sent for any specific data node, client/management applications can not rely on the on-change functionality unless the client has some means to identify for which objects on-change notifications are supported and for which ones they are not supported. YANG models are meant to be used as an interface contract. Without identification of data nodes supporting on-change, today this contract only states the YANG server may (or may not) send on-change notifications for a data node specified in a YANG module.

This document proposes a YANG module that allows a client to identify which data nodes support on-change notification, removing the uncertainty for on-change notifications.

On-change Notification Capability information will be needed both in implementation-time and run-time.

Run-time information is needed

- for any "purely model driven" client
- to check that early information about the capability is indeed what the server supports
- in case the capability might change during run-time e.g. due to licensing, HW constraints etc.

Implementation time information is needed by Network Management System (NMS) implementers. During NMS implementation for any functionality that depends on the notifications the information about on change notification capability is needed. If the information is not available early in some document, but only as instance data from the network node, the NMS implementation will be delayed, because it has to wait for the network node to be ready. Also assuming that all NMS implementers will have a correctly configured network node available to retrieve data from, is very expensive. (An NMS may handle dozens of network node types.) Often the NMS is a requirement for introducing a new network node type into a network, so delaying the NMS effectively delays the availability of the network node as well.

Implementation time information is needed by system integrators. System integrators will need information about on change notification capability early. When introducing a network node type into their network Network operators often need to integrate the node type into their own management system. The NMS may have management functions that depend on on-change notifications. The network operator needs to plan his management practices and NMS implementation before he even decides to buy the specific network node type. Moreover the decision to buy the node type sometimes depends on these management possibilities.

3. On-change Notification Capability Model

As described above a number of stakeholders need information about the on change notification capability both in implementation and run-time. It is a goal to provide this information in a format that is

- vendor independent (standard)

- formal (no freeform English text please)

- the same both in implementation-time and run-time

The YANG module ietf-notification-capabilities is defined to provide information about the on-change notification capabilities. It is defined as an extension to the ietf-yang-library module [I-D.ietf-netconf-rfc7895bis]. For each YANG Module listed in the ietf-yang-library there is a default notification capability separately for config false and config true data nodes. There is also an on-change-notification-capability list containing a potentially different true/false notification capability for any data nodes in the schema tree. Unless a node is in the list with a specific capability value, it inherits its on-change-notification-capability from its parent in the data tree, or from the relevant default values.

The instance information can be provided in one of two ways:

- It can be dynamically populated by a server during runtime.

- It can be provided by an implementer as YANG instance data (possibly retrievable through NETCONF or RESTCONF, but preferably also through other means including but not limited to download from product Websites) according to [I-D.lengyel-netmod-yang-instance-data]

As the same information is also needed in implementation-time YANG servers SHOULD document their on-change-notification-capabilities using YANG Instance data according to [I-D.lengyel-netmod-yang-instance-data] following the ietf-notification-capabilities module.

3.1. Tree Diagram

The following tree diagram [I-D.ietf-netmod-yang-tree-diagrams] provides an overview of the data model.

```
module: ietf-notification-capabilities
  augment /yanglib:yang-library/yanglib:module-set:
    +--ro notification-sent-for-config-default?   boolean
    +--ro notification-sent-for-state-default?    boolean
    +--ro on-change-notification-capability* [data-node-selector]
    |  +--ro data-node-selector    nacm:node-instance-identifier
    +--ro on-change-notification-sent?            boolean
```

3.2. YANG Module

```
   <CODE BEGINS> file "ietf-notification-capabilities.yang"

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

     import ietf-yang-library {
       prefix yanglib;
       reference "rfc7895bis";
     }

     import ietf-netconf-acm { prefix nacm; }

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

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

        WG Chair: Kent Watsen
                  <mailto:kwatsen@juniper.net>

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

        Editor:   Balazs Lengyel
                  <mailto:balazs.lengyel@ericsson.com>";


     description "This module augments the ietf-yang-library to
       specify per module for which data nodes will the YANG server
       send on-change notifications.

       On-change notification capability is marked as true or false.
       This marking is inherited from the parent down the data tree
       unless explicitly marked otherwise.

       On-change notifications SHALL be sent for a configuration
       data node if one of the following is true:
       - it is specified in the on-change-notification-capability
       list and has a on-change-notification-sent value true or
       - notifications are sent for its parent data node and it is
       not specified in the on-change-notification-capability list or
```

```
          - if it is a top level data-node and is not specified in the
          on-change-notification-capability list and the
          notification-sent-for-config-default is true.

          For state data nodes the same rules appiy except for top
          level nodes the notification-sent-for-state-default value
          SHALL be considered.
          ";

        revision 2018-02-20 {
          description "Initial version";
          reference
            "RFC XXX: YangPush Notification Capabilities";
        }

        // if we base on the RFC7895 current version
        // augment /yanglib:modules-state/yanglib:modules/yanglib:module {

        // if we base on the RFC7895bis
        augment /yanglib:yang-library/yanglib:module-set {
          description "Extends the ietf-yang-library module with
              on-change-notification-capability information";

          leaf notification-sent-for-config-default {
            type boolean;
            default true;
            description "Specifies the default value for this module’s
              top level configuration data nodes for the
              on-change-notification-sent capability.";
          }

          leaf notification-sent-for-state-default {
            type boolean;
            default false;
            description "Specifies the default value for this module’s
              top level state data nodes for the
              on-change-notification-sent capability.";
          }

          list on-change-notification-capability {
            key data-node-selector ;
            description "A list of data nodes that have the
               on-change-notification-capability specifically defined.";

            leaf data-node-selector {
              type nacm:node-instance-identifier;
              description data-node-selector;
              }
```

```
         }

         leaf on-change-notification-sent {
           type boolean;
           description "Specifies whether the YANG server will
             send on-change notifications for the selected data nodes.";
       }
     }
   }

   <CODE ENDS>
```

4. Security Considerations

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

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

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 registrations are requested:

```
      URI: urn:ietf:params:xml:ns:yang:ietf-notification-capabilities
      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 [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

```
  name:       ietf-notification-capabilities
  namespace:  urn:ietf:params:xml:ns:yang:ietf-notification-capabilities
  prefix:     inc
  reference:  RFC XXXX
```

6. References

6.1. Normative References

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-04 (work in progress), January 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-14 (work in progress), February 2018.

[I-D.lengyel-netmod-yang-instance-data]
Lengyel, B. and B. Claise, "YANG Instance Data Files and their use for Documenting Server Capabilities", draft-lengyel-netmod-yang-instance-data-00 (work in progress), February 2018.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

6.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Authors’ Addresses

Balazs Lengyel
Ericsson
Magyar Tudosok korutja 11
1117 Budapest
Hungary

Email: balazs.lengyel@ericsson.com

Alexander Clemm
Huawei USA
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ludwig@clemm.org

NETCONF WG
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2018

M. Jethanandani
J. Lam
A. Leung
Cisco Systems, Inc.

February 23, 2018

Binary Encoding for NETCONF
draft-mahesh-netconf-binary-encoding-00

Abstract

This document describes a method by which a NETCONF [RFC6241] client and server can negotiate an alternate form of encoding.

This document updates RFC 6241.

Status of This Memo

This Internet-Draft will expire on August 27, 2018.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241], by default, supports XML encoding for its payload. However, XML can be very verbose, specially for operational data. That combined with parsing of tags leads to slow processing of the data. This document proposes a mechanism by which clients and servers can negotiate an alternate form of encoding, e.g. binary encoding, and use that to exchange data. Binary encoding can reduce the physical size of the data exchanged, in some cases by as much as 66%, and improve the time that is required to process the data, while preserving the original data.

Several binary encoding mechanisms have been proposed, including CBOR [RFC7049]. This document does not advocate any particular binary encoding format. Instead, it leaves it up to the negotiation between client and server to decide the form of encoding. For an example of how to encode YANG in CBOR format, see CBOR Encoding of Data Modeled with YANG [I-D.ietf-core-yang-cbor].

1.1. Definitions and Acronyms

2. Protocol Negotiation

NETCONF clients and servers exchange a hello as part of establishing a connection. As part of the hello exchange, each of them advertises their set of capabilities. This draft suggests advertisement of the following additional capability.

2.1. Encoding

2.1.1. Overview

The :encoding capability indicates what encoding format each side is willing to support. If the client and server are capable of supporting multiple forms of encoding, they can list each of them. There is no need to include xml in the list, as that is supported by default.

2.1.2. Dependencies

When using this capability, any binardy encoding needs the underlying transport to be 8-bit clean, and which preserves message boundaries when dealing with arbitrary binary data. This requires use of Chunked Framing mechanism as described in NETCONF over SSH [RFC6242].

2.1.3. Capability Identifier

The :encoding capability is identfied by the following capability string:

```
urn:ietf:params:netconf:capability:encoding:1.0?format={name, ...}
```

The :encoding capability URI MUST contain a "format" argument assigned a comma-separated list of formats supported by the device. For example:

```
urn:ietf:params:netconf:capability:encoding:1.0?format=cbor,gpb,thrif
t
```

2.1.4. New Operation

2.1.4.1. <activate>

Description:

After each side has exchanged capabilities, a client can initiate a request to switch to a new encoding format using the <activate> operation.

Parameters:

encoding:

The <activate> operation instructs the server to switch to the new binary format. If the server does not support the new binary format or is unable to switch to the new binary format for any reason, it MUST fail with the <error-tag> value of "not-supported" and keep the existing encoding format it is using.

If the system does not have the :encoding capability, the <activate> operation is not available. If there is a desire to fall back to default encoding of XML, the client needs to terminate the existing connection and establish a new connection.

Positive Response:

If the device is able to satisfy the requests, an <rpc-reply> is sent that contains an <ok> element.

Negative Response:

An <rpc-error> element is included in the <rpc-reply> with the <type> element set to "not-supported". The <error-tag> element must be set to "server-error".

Example:

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <activate>
    <encoding>cbor</encoding>
  </activate>
</rpc>

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>
```

3. Security Considerations

4. IANA Considerations

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

4.1. NETCONF Capability URNs

IANA registry "Network Configuration Protocol (NETCONF) Capability URNs" needs to be updated to include the following capability.

```
Index
    Capability Identifier
-------------------------
:encoding
    urn:ietf:params:netconf:capability:encoding:1.0
```

5. Acknowledgements

6. References

6.1. Normative References

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

6.2. Informative References

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-06 (work in progress), February 2018.

Authors’ Addresses

Mahesh Jethanandani

Email: mjethanandani@gmail.com

Jason Lam
Cisco Systems, Inc.

Email: lamj@cisco.com

Alfred Leung
Cisco Systems, Inc.

Email: alfleung@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

R. Shakir
A. Shaikh
P. Borman
M. Hines
C. Lebsack
C. Morrow
Google

March 13, 2017

gRPC Network Management Interface (gNMI)
draft-openconfig-rtgwg-gnmi-spec-00

Abstract

This document describes the gRPC Network Management Interface (gNMI), a network management protocol based on the gRPC RPC framework. gNMI supports retrieval and manipuation of state from network elements where the data is represented by a tree structure, and addressable by paths. The gNMI service defines operations for configuration management, operational state retrieval, and bulk data collection via streaming telemtry.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a gRPC [1]-based protocol for the modification and retrieval of configuration from a network element, as well as the control and generation of telemetry streams from a network element to a data collection system. The intention is that a single gRPC service definition can cover both configuration and telemetry - allowing a single implementation on the network element, as well as a single NMS element to interact with the device via telemetry and configuration RPCs.

All messages within the gRPC service definition are defined as protocol buffers [2] (specifically proto3). gRPC service definitions are expected to be described using the relevant features of the protobuf IDL. A reference protobuf definition is maintained in [REFERENCE-PROTO] [3]. The current, authoritative version of this specification is available at [GNMI-SPEC] [4].

The service defined within this document is assumed to carry payloads that contain data instances of OpenConfig [5] YANG schemas, but can be used for any data with the following characteristics:

1. structure can be represented by a tree structure where nodes can be uniquely identified by a path consisting of node names, or node names coupled with attributes;

2. values can be serialised into a scalar object.

Currently, values may be serialised to a scalar object through encoding as a JSON string, a byte-array, or a serialised protobuf object - although the definition of new serialisations is possible.

Throughout this specification the following terminology is used:

- _Telemetry_ - refers to streaming data relating to underlying characteristics of the device - either operational state or configuration.

- _Configuration_ - elements within the data schema which are read/write and can be manipulated by the client.

- _Target_ - the device within the protocol which acts as the owner of the data that is being manipulated or reported on. Typically this will be a network device.

- _Client_ - the device or system using the protocol described in this document to query/modify data on the target, or act as a collector for streamed data. Typically this will be a network management system.

2. Common Message Types and Encodings

2.1. Reusable Notification Message Format

When a target wishes to communicate data relating to the state of its internal database to an interested client, it does so via means of a common "Notification" message. Notification messages are reused in other higher-layer messages for various purposes. The exact use of the Notification message is described on a per-RPC basis.

The fields of the Notification message are as follows:

- "timestamp" - The time at which the data was collected by the device from the underlying source, or the time that the target generated the Notification message (in the case that the data does not reflect an underlying data source). This value is always represented according to the definition in Section 2.2.1.

- "prefix" - a prefix which is applied to all path fields (encoded as per Section 2.2.2) included in the "Notification" message. The paths expressed within the message are formed by the concatenation of "prefix + path". The "prefix" always precedes the "path" elements. Further semantics of prefixes are described in Section 2.4.1.

- "alias"- a string providing an alias for the prefix specified within the notification message. The encoding of an alias, and the procedure for their creation is described in Section 2.4.2"."

- "update" - a list of update messages that indicate changes in the underlying data of the target. Both modification and creation of data is expressed through the update message.

  - An "Update" message has two subfields:

    - "path" - a path encoded as per Section 2.2.2.

    - "value" - a value encoded as per Section 2.2.3.

  - The set of paths that are specified within the list of updates MUST be unique. In this context, the path is defined to be the fully resolved path (including the prefix). In the case that there is a duplicate path specified within an update, only the final update should be processed by the receiving entity.

- "delete" - a list of paths (encoded as per Section 2.2.2) that indicate the deletion of data nodes on the target.

The creator of a Notification message MUST include the "timestamp" field. All other fields are optional.

2.2. Common Data Types

2.2.1. Timestamps

Timestamp values MUST be represented as the number of nanoseconds since the Unix epoch (January 1st 1970 00:00:00 UTC). The value MUST be encoded as a signed 64-bit integer ("int64").

2.2.2. Paths

Paths are represented according to gNMI Path Conventions [6], a simplified form of XPATH. Rather than utilising a single string to represent the path - with the "/" character separating each element of the path, the path is represented by an ordered list of strings, starting at the root node, and ending at the most specific path element.

A path is represented by the "Path" message with the following fields:

- "element" -- a set of path elements, encoded as strings (see examples below).

- "origin" - field which MAY be used to disambiguate the path if necessary. For example, the origin may be used to indicate which organization defined the schema to which the path belongs.

Each "Path" element should correspond to a node in the data tree. For example, the path "/a/b/c/d" is encoded as:

```
path: <
  element: "a"
  element: "b"
  element: "c"
  element: "d"
>
```

Where attributes are to be specified, these are encoded alongside the node name within the path element, for example a node specified by "/a/e[key=k1]/f/g" would have the path encoded as:

```
path: <
  element: "a"
  element: "e[key=k1]"
  element: "f"
  element: "g"
>
```

The root node ("/") is indicated by encoding a single path element which is an empty string, as per the following example:

```
path: <
  element: ""
>
```

Paths (defined to be the concatenation of the "Prefix" and "Path" within the message) specified within a message MUST be absolute - no messages with relative paths should be generated.

2.2.3. Node Values

The value of a data node is encoded as a two-field message:

- "bytes" - an arbitrary series of bytes which indicates the value of the node referred to within the message context.

- "type" - a field indicating the type of data contained in the bytes field. Currently defined types are:

2.3. Encoding Data in an Update Message

2.3.1. JSON and JSON_IETF

The "JSON" type indicates that the value included within the "bytes" field of the node value message is encoded as a JSON string. This format utilises the specification in RFC7159 [7]. Additional types (e.g., "JSON_IETF") are utilised to indicate specific additional characteristics of the encoding of the JSON data (particularly where they relate to serialisation of YANG-modeled data).

For any JSON encoding:

- In the case that the data item at the specified path is a leaf node (i.e., has no children) the value of that leaf is encoded directly - i.e., the "bare" value is specified (i.e., a JSON object is not required, and a bare JSON value is included).

- Where the data item referred to has child nodes, the value field contains a serialised JSON entity (object or array) corresponding to the referenced item.

Using the following example data tree:

```
               root +
                    |
                    +-- a +
                          |
                          +-- b[name=b1] +
                                         |
                                         +-- c +
                                               |
                                               +-- d (string)
                                               +-- e (uint32)
```

The following serialisations would be used (note that the examples below follow the conventions for textproto, and Golang-style backticks are used for string literals that would otherwise require escaping):

For "/a/b[name=b1]/c/d":

```
                         update: <
                           path: <
                             element: "a"
                             element: "b[name=b1]"
                             element: "c"
                             element: "d"
                           >
                           value: <
                             value: "AStringValue"
                             type: JSON
                           >
                         >
```

For "/a/b[name=b1]/c/e":

```
                 update: <
                   path: <
                     element: "a"
                     element: "b[name=b1]"
                     element: "c"
                     element: "e"
                   >
                   value: <
                     Value: 10042    // decoded byte array
                     type: JSON
                   >
                 >
```

For "/a/b[name=b1]/c":

```
              update: <
                path: <
                  element: "a"
                  element: "b[name=b1]"
                  element: "c"
                >
                value: <
                  value: { "d": "AStringValue", "e": 10042 }
                  type: JSON
                >
              >
```

For "/a" :

```
               update: <
                 path: <
                   element: "a"
                 >
                 value: <
                   value: ‘{ "b": [
                                     {
                                       "name": "b1",
                                       "c": {
                                       "d": "AStringValue",
                                         "e": 10042
                                       }
                                      }
                                  ]
                           }‘
                   type: JSON_IETF
                 >
               >
```

Note that all JSON values MUST be valid JSON. That is to say, whilst a value or object may be included in the message, the relevant quoting according to the JSON specification in RFC7159 [8] must be used. This results in quoted string values, and unquoted number values.

"JSON_IETF" encoded data MUST conform with the rules for JSON serialisation described in RFC7951 [9]. Data specified with a type of JSON MUST be valid JSON, but no additional constraints are placed upon it. An implementation MUST NOT serialise data with mixed "JSON" and "JSON_IETF" encodings.

Both the client and target MUST support the JSON encoding as a minimum.

2.3.2. Bytes

The "BYTES" type indicates that the contents of the "bytes" field of the message contains a byte sequence whose semantics is opaque to the protocol.

2.3.3. Protobuf

The "PROTOBUF" type indicates that the contents of the "bytes" field of the message contains a serialised protobuf message. Note that in the case that the sender utilises this type, the receiver must understand the schema (and hence the type of protobuf message that is serialised) in order to decode the value. Such agreement is not guaranteed by the protocol and hence must be established out-of-band.

2.3.4. ASCII

The "ASCII" type indicates that the contents of the "bytes" field of the message contains system-formatted ASCII encoded text. For configuration data, for example, this may consist of semi-structured CLI configuration data formatted according to the target platform. The gNMI protocol does not define the format of the text - this must be established out-of-band.

2.4. Use of Data Schema Paths

2.4.1. Path Prefixes

In a number of messages, a prefix can be specified to reduce the lengths of path fields within the message. In this case, a "prefix" field is specified within a message - comprising of a valid path encoded according to Section Section 2.2.2 In the case that a prefix is specified, the absolute path is comprised of the concatenation of the list of path elements representing the prefix and the list of path elements in the "path" field.

For example, again considering the data tree shown in Section Section 2.3.1 if a "Notification" message updating values, a prefix could be used to refer to the "/a/b[name=b1]/c/d" and "/a/b[name=b1]/c/e"data nodes:

```
     notification: <
         timestamp: (timestamp)      // timestamp as int64
         prefix: <
          element: "a"
           element: "b[name=b1]"
           element: "c"
         >
       update: <
         path: <
           element: "d"
         >
         value: <
           value: "AStringValue"
           type: JSON
         >
       >
       update: <
         path: <
           element: "e"
         >
         value: <
           value: 10042      // converted to int representation
           type: JSON
         >
       >
     >
```

2.4.2. Path Aliases

In some cases, a client or target MAY desire to utilises aliases for a particular path - such that subsequent messages can be compressed by utilising the alias, rather than using a complete representation of the path. Doing so reduces total message length, by ensuring that redundant information can be removed.

Support for path aliases MAY be provided by a target. In a case where a target does not support aliases, the maximum message length SHOULD be considered, especially in terms of bandwidth utilisation, and the efficiency of message generation.

A path alias is encoded as a string. In order to avoid valid data paths clashing with aliases (e.g., "a" in the above example), an alias name MUST be prefixed with a "#" character.

The means by which an alias is created is defined on a per-RPC basis. In order to delete an alias, the alias name is sent with the path corresponding to the alias empty.

Aliases MUST be specified as a fully expanded path, and hence MUST NOT reference other aliases within their definition, such that a single alias lookup is sufficient to resolve the absolute path.

2.4.3. Interpretation of Paths Used in RPCs

When a client specifies a path within an RPC message which indicates a read, or retrieval of data, the path MUST be interpreted such that it refers to the node directly corresponding with the path *and* all its children. The path refers to the direct node and all descendent branches which originate from the node, recursively down to each leaf element. If specific nodes are expected to be excluded then an RPC MAY provide means to filter nodes, such as regular-expression based filtering, lists of excluded paths, or metadata-based filtering (based on annotations of the data schema being manipulated, should such annotations be available and understood by both client and target).

For example, consider the following data tree:

```
                 root +
                      |
                      +-- childA +
                      |          |
                      |          +-- leafA1
                      |          +-- leafA2
                      |          +-- childA3 --+
                      |                        |
                      |                        +-- leafA31
                      |                        +-- leafA32
                      |
                      +-- childB +
                                 |
                                 +-- leafB1
                                 +-- leafB2
```

A path referring to "root" (which is represented by a Path consisting of a single element specifying an empty string) should result in the nodes "childA" and "childB" and all of their children ("leafA1, leafA2, leafB1, leafB2, childA3, leafA31" and "leafA32") being considered by the relevant operation.

In the case that the RPC is modifying the state of data (i.e., a write operation), such recursion is not required - rather the modification operation should be considered to be targeted at the node within the schema that is specified by the path, and the value should be deserialized such that it modifies the content of any child nodes if required to do so.

2.5. Error handling

Where the client or target wishes to indicate an error, an "Error" message is generated. Errors MUST be represented by a canonical gRPC error code (Java [10], Golang [11], C++ [12]) . The entity generating the error MUST specify a free-text string which indicates the context of the error, allowing the receiving entity to generate log entries that allow a human operator to understand the exact error that occurred, and its context. Each RPC defines the meaning of the relevant canonical error codes within the context of the operation it performs.

The canonical error code that is chosen MUST consider the expected behavior of the client on receipt of the message. For example, error codes which indicate that a client may subsequently retry SHOULD only be used where retrying the RPC is expected to result in a different outcome.

A re-usable "Error" message MUST be used when sending errors in response to an RPC operation. This message has the following fields:

- "code" - an unsigned 32-bit integer value corresponding to the canonical gRPC error code.

- "message"- a human-readable string describing the error condition in more detail. This string is not expected to be machine-parsable, but rather provide contextual information which may be passed to upstream systems.

- "data"- an arbitrary sequence of bytes (encoded as "[proto.Any](https://github.com/google/protobuf/blob/master/src/google/protobuf/any.proto)") which provides further contextual information relating to the error.

2.6. Schema Definition Models

The data tree supported by the target is expected to be defined by a set of schemas. The definition and format of these models is out of scope of this specification (YANG-modeled data is one example). In the case that such schema definitions are used, the client should be able to determine the models that are supported by the target, so that it can generate valid modifications to the data tree, and interpret the data returned by "Get" and "Subscribe" RPC calls.

Additionally, the client may wish to restrict the set of models that are utilised by the target so that it can validate the data returned to it against a specific set of data models. This is particularly relevant where the target may otherwise add new values to restricted value data elements (e.g., those representing an enumerated type), or augment new data elements into the data tree.

In order to allow the client to restrict the set of data models to be used when interacting with the target, the client MAY discover the set of models that are supported by the target using the "Capabilities" RPC described in Section 3.2. For subsequent "Get" and "Subscribe" RPCs, the client MAY specify the models to be used by the target. The set of models to use is expressed as a "ModelData" message, as specified in Section 2.6.1.

If the client specifies a set of models in a "Get" or "Subscribe" RPC, the target MUST NOT utilize data tree elements that are defined in schema modules outside the specified set. In addition, where there are data tree elements that have restricted value sets (e.g., enumerated types), and the set is extended by a module which is outside of the set, such values MUST NOT be used in data instances that are sent to the client. Where there are other elements of the schema that depend on the existence of such enumerated values, the target MUST NOT include such values in data instances sent to the client.

2.6.1. The ModelData message

The "ModelData" message describes a specific model that is supported by the target and used by the client. The fields of the "ModelData" message identify a data model registered in a model catalog, as described in [MODEL_CATALOG_DOC] [13] (the schema of the catalog itself - expressed in YANG - is described in [MODEL_CATALOG_YANG [14]]). Each model specified by a "ModelData" message may refer to a specific schema module, a bundle of modules, or an augmentation or deviation, as described by the catalog entry.

Each "ModelData" message contains the following fields:

- "name" - name of the model expressed as a string.
- "organization" - the organization publishing the model, expressed as a string.
- "version" - the supported (or requested) version of the model, expressed as a string which represents the semantic version of the catalog entry.

The combination of "name", "organization", and "version" uniquely identifies an entry in the model catalog.

3. Service Definition

A single gRPC service is defined - future revisions of this specification MAY result in additional services being introduced, and hence an implementation MUST NOT make assumptions that limit to a single service definition.

The service consists of the following RPCs:

- "Capabilities" - defined in Section 3.2 and used by the client and target as an initial handshake to exchange capability information
- "Get" - defined in Section 3.3, used to retrieve snapshots of the data on the target by the client.
- "Set" - defined in Section 3.4 and used by the client to modify the state of the target.
- "Subscribe" - defined in Section 3.5 and used to control subscriptions to data on the target by the client.

3.1. Session Security, Authentication and RPC Authorization

The session between the client and server MUST be encrypted using TLS - and a target or client MUST NOT fall back to unencrypted channels.

New connections are mutually authenticated -- each entity validates the X.509 certificate of the remote entity to ensure that the remote entity is both known, and authorized to connect to the local system.

If the target is expected to authenticate an RPC operation, the client MUST supply a username and password in the metadata of the RPC message (e.g., "SubscribeRequest", "GetRequest" or "SetRequest"). If the client supplies username/password credentials, the target MUST authenticate the RPC per its local authentication functionality.

Authorization is also performed per-RPC by the server, through validating client-provided metadata. The client MAY include the appropriate AAA metadata, which MUST contain a username, and MAY include a password in the context of each RPC call it generates. If the client includes both username and password, the target MUST authenticate and authorize the request. If the client only supplies the username, the target MUST authorize the RPC request.

A more detailed discussion of the requirements for authentication and encryption used for gNMI is in [GNMI-AUTH] [15].

3.2. Capability Discovery

A client MAY discover the capabilities of the target using the "Capabilities" RPC. The "CapabilityRequest" message is sent by the client to interrogate the target. The target MUST reply with a "CapabilityResponse" message that includes its gNMI service version, the versioned data models it supports, and the supported data encodings. This information is used in subsequent RPC messages from the client to indicate the set of models that the client will use (for "Get", "Subscribe" as described in Section 2.6) , and the encoding to be used for the data.

When the client does not specify the models it is using, the target SHOULD use all data schema modules that it supports when considering the data tree to be addressed. If the client does not specify the encoding in an RPC message, it MUST send JSON encoded values (the default encoding).

3.2.1. The CapabilityRequest message

The "CapabilityRequest" message is sent by the client to request capability information from the target. The "CapabilityRequest" message carries no additional fields.

3.2.2. The CapabilityResponse message

The "CapabilityResponse" message has the following fields:

- o "supported_models" - a set of "ModelData" messages (as defined in Section 2.6.1) describing each of the models supported by the target

- o "supported_encodings" - an enumeration field describing the data encodings supported by the target, as described in Section 2.3.

- "gNMI_version" - the semantic version of the gNMI service supported by the target, specified as a string. The version should be interpreted as per [OPENCONFIG-SEMVER [16]].

3.3. Retrieving Snapshots of State Information

In some cases, a client may require a snapshot of the state that exists on the target. In such cases, a client desires some subtree of the data tree to be serialized by the target and transmitted to it. It is expected that the values that are retrieved (whether writeable by the client or not) are collected immediately and provided to the client.

The "Get" RPC provides an interface by which a client can request a set of paths to be serialized and transmitted to it by the target. The client sends a "GetRequest" message to the target, specifying the data that is to be retrieved. The fields of the "GetRequest" message are described in Section 3.3.1.

Upon reception of a "GetRequest", the target serializes the requested paths, and returns a "GetResponse" message. The target MUST reflect the values of the specified leaves at a particular collection time, which MAY be different for each path specified within the "GetRequest" message.

The target closes the channel established by the "Get" RPC following the transmission of the "GetResponse" message.

3.3.1. The GetRequest Message

The "GetRequest" message contains the following fields:

- "prefix" - a path (specified as per Section 2.2.2), and used as described in Section 2.4.1. The prefix is applied to all paths within the "GetRequest" message.

- "path" - a set of paths (expressed as per Section 2.2.2) for which the client is requesting a data snapshot from the target. The path specified MAY utilize wildcards. In the case that the path specified is not valid, the target MUST populate the "error" field of the "GetResponse" message indicating an error code of "InvalidArgument" and SHOULD provide information about the invalid path in the error message.

- "type" - the type of data that is requested from the target. The valid values for type are described below.

- "encoding" - the encoding that the target should utilise to serialise the subtree of the data tree requested. The type MUST be one of the encodings specified in Section 2.3. If the "Capabilities" RPC has been utilised, the client SHOULD use an encoding advertised as supported by the target. If the encoding is not specified, JSON MUST be used. If the target does not support the specified encoding, the target MUST populate the error field of the "GetResponse" message, specifying an error of "InvalidArgument". The error message MUST indicate that the specified encoding is unsupported.

- "use_models" - a set of "ModelData" messages (defined in Section 2.6.1) indicating the schema definition modules that define the data elements that should be returned in response to the Get RPC call. The semantics of the "use_models" field are defined in Section 2.6.

Since the data tree stored by the target may consist of different types of data (e.g., values that are operational in nature, such as protocol statistics) - the client MAY specify that a subset of values in the tree are of interest. In order for such filtering to be implemented, the data schema on the target MUST be annotated in a manner which specifies the type of data for individual leaves, or subtrees of the data tree.

The types of data currently defined are:

- "CONFIG" - specified to be data that the target considers to be read/write. If the data schema is described in YANG, this corresponds to the "config true" set of leaves on the target.

- "STATE" - specified to be the read-only data on the target. If the data schema is described in YANG, "STATE" data is the "config false" set of leaves on the target.

- "OPERATIONAL" - specified to be the read-only data on the target that is related to software processes operating on the device, or external interactions of the device.

If the "type" field is not specified, the target MUST return CONFIG, STATE and OPERATIONAL data fields in the tree resulting from the client’s query.

3.3.2. The GetResponse message

The "GetResponse" message consists of:

- "notification" - a set of "Notification" messages, as defined in Section 2.1. The target MUST generate a "Notification" message for each path specified in the client's "GetRequest", and hence MUST NOT collapse data from multiple paths into a single "Notification" within the response. The "timestamp" field of the "Notification" message MUST be set to the time at which the target's snapshot of the relevant path was taken.

- "error" - an "Error" message encoded as per the specification in Section 2.5, used to indicate errors in the "GetRequest" received by the target from the client.

3.3.3. Considerations for using Get

The "Get" RPC is intended for clients to retrieve relatively small sets of data as complete objects, for example a part of the configuration. Such requests are not expected to put a significant resource burden on the target. Since the target is expected to return the entire snapshot in the "GetResponse" message, "Get" is not well-suited for retrieving very large data sets, such as the full contents of the routing table, or the entire component inventory. For such operations, the "Subscribe" RPC is the recommended mechanism, e.g. using the "ONCE" mode as described in Section 3.5.

Another consideration for "Get" is that the timestamp returned is associated with entire set of data requested, although individual data items may have been sampled by the target at different times. If the client requires higher accuracy for individual data items, the "Subscribe" RPC is recommended to request a telemetry stream (see Section 3.5.2).

3.4. Modifying State

Modifications to the state of the target are made through the "Set" RPC. A client sends a "SetRequest" message to the target indicating the modifications it desires.

A target receiving a "SetRequest" message processes the operations specified within it - which are treated as a transaction (see Section 3.4.3). The server MUST process deleted paths (within the "delete" field of the "SetRequest"), followed by paths to be replaced (within the "replace" field), and finally updated paths (within the "update" field). The order of the replace and update fields MUST be treated as significant within a single "SetRequest" message. If a single path is specified multiple times for a single operation (i.e., within "update" or "replace"), then the state of the target MUST reflect the application of all of the operations in order, even if they overwrite each other.

In response to a "SetRequest", the target MUST respond with a "SetResponse" message. For each operation specified in the "SetRequest" message, an "UpdateResult" message MUST be included in the response field of the "SetResponse". The order in which the operations are applied MUST be maintained such that "UpdateResult" messages can be correlated to the "SetRequest" operations. In the case of a failure of an operation, the "error" field of the "UpdateResult" message MUST be populated with an "Error" message as per the specification in Section 2.5. In addition, the "error" field of the "SetResponse" message MUST be populated with an error message indicating the success or failure of the set of operations within the "SetRequest" message (again using the error handling behavior defined in Section 2.5).

3.4.1. The SetRequest Message

A "SetRequest" message consists of the following fields:

- "prefix" - specified as per Section 2.4.1. The prefix specified is applied to all paths defined within other fields of the message.

- "delete" - A set of paths, specified as per Section 2.2.2, which are to be removed from the data tree. A specification of the behavior of a delete is defined in Section 3.4.6.

- "replace" - A set of "Update" messages indicating elements of the data tree whose content is to be replaced.

- "update" - A set of "Update" messages indicating elements of the data tree whose content is to be updated.

The semantics of "updating" versus "replacing" content are defined in Section 3.4.4

A re-usable "Update" message is utilised to indicate changes to paths where a new value is required. The "Update" message contains two fields:

- "path" - a path encoded as per Section 2.2.2 indicating the path of the element to be modified.

- "value" - a value encoded as per Section 2.2.3 indicating the value applied to the specified node. The semantics of how the node is updated is dependent upon the context of the update message, as specified in Section 3.4.4.

3.4.2. The SetResponse Message

A "SetResponse" consists of the following fields:

- "prefix" - specified as per Section 2.4.1. The prefix specified is applied to all paths defined within other fields of the message.

- "message" - an error message as specified in Section 2.5. The target SHOULD specify a "message" in the case that the update was successfully applied, in which case an error code of "OK (0)" "MUST" be specified. In cases where an update was not successfully applied, the contents of the error message MUST be specified as per Section 2.5.

- "response" - containing a list of responses, one per operation specified within the "SetRequest" message. Each response consists of an "UpdateResult" message with the following fields:

  * "timestamp" - a timestamp (encoded as per Section 2.2.1) at which the set request message was accepted by the system.

  * "path" - the path (encoded as per Section 2.2.2) specified within the "SetRequest". In the case that a common prefix was not used within the "SetRequest", the target MAY specify a "prefix" to reduce repetition of path elements within multiple "UpdateResult" messages in the "request" field.

  * "op" - the operation corresponding to the path. This value MUST be one of "DELETE", "REPLACE", or "UPDATE".

  * "message" - an error message (as specified in Section 2.5). This field follows the same rules as the message field within the "SetResponse" message specified above.

3.4.3. Transactions

All changes to the state of the target that are included in an individual "SetRequest" message are considered part of a transaction. That is, either all modifications within the request are applied, or the target MUST rollback the state changes to reflect its state before any changes were applied. The state of the target MUST NOT appear to be changed until such time as all changes have been accepted successfully. Hence, telemetry update messages MUST NOT reflect a change in state until such time as the intended modifications have been accepted.

As per the specification in Section 3.4, within an individual transaction ("SetRequest") the order of operations is "delete", "replace", "update".

As the scope of a "transaction" is a single "SetRequest" message, a client desiring a set of changes to be applied together MUST ensure that they are encapsulated within a single "SetRequest" message.

3.4.4. Modes of Update: Replace versus Update

Changes to read-write values on the target are applied based on the "replace" and "update" fields of the "SetRequest" message.

For both replace and update operations, if the path specified does not exist, the target MUST create the data tree element and populate it with the data in the "Update" message, provided the path is valid according to the data tree schema. If invalid values are specified, the target MUST cease processing updates within the "SetRequest" method, return the data tree to the state prior to any changes, and return a "SetResponse" message indicating the error encountered.

For "replace" operations, the behavior regarding omitted data elements in the "Update" depends on whether they refer to non-default values (i.e., set by a previous "SetRequest"), or unmodified defaults. When the "replace" operation omits values that have been previously set, they MUST be treated as deleted from the data tree. Otherwise, omitted data elements MUST be created with their default values on the target.

For "update" operations, only the value of those data elements that are specified explicitly should be treated as changed.

3.4.5. Modifying Paths Identified by Attributes

The path convention defined in Section 2.2.2 allows nodes in the data tree to be identified by a unique set of node names (e.g.,"/a/b/c/d") or paths that consist of node names coupled with attributes (e.g., "/a/e[key=10]"). In the case where where a node name plus attribute name is required to uniquely identify an element (i.e., the path within the schema represents a list, map, or array), the following considerations apply:

- In the case that multiple attribute values are required to uniquely address an element - e.g., "/a/f[k1=10][k2=20]"- and a replace or update operation’s path specifies a subset of the attributes (e.g., "/a/f[k1=10]") then this MUST be considered an error by the target system - and an error code of"InvalidArgument (3)" specified.

- o Where the path specified refers to a node which itself represents the collection of objects (list, map, or array) a replace operation MUST remove all collection entries that are not supplied in the value provided in the "SetRequest". An update operation MUST be considered to add new entries to the collection if they do not exist.

- o In the case that key values are specified both as attributes of a node, and as their own elements within the data tree, update or replace operations that modify instances of the key in conflicting ways MUST be considered an error. The target MUST return an error code of "InvalidArgument (3)".

For example, consider a tree corresponding to the examples above, as illustrated below.

```
           root +
                |
                + a --+
                        |
                        +-- f[k1=10][k2=20] --+
                        |                     |
                        |                     +-- k1 = 10
                        |                     +-- k2 = 20
                        |
                        +-- f[k1=10][k2=21] --+
                                              |
                                              +-- k1 = 10
                                              +-- k2 = 21
```

In this case, nodes "k1" and "k2" are standalone nodes within the schema, but also correspond to attribute values for the node ""f"". In this case, an update or replace message specifying a path of "/a/f[k1=10][k2=20]" setting the value of "k1" to 100 MUST be considered erroneous, and an error code of "InvalidArgument (3)" specified.

3.4.6. Deleting Configuration

Where a path is contained within the "delete" field of the "SetRequest" message, it should be removed from the target’s data tree. In the case that the path specified is to an element that has children, these children MUST be recursively deleted. If a wildcard path is utilised, the wildcards MUST be expanded by the target, and the corresponding elements of the data tree deleted. Such wildcards MUST support paths specifying a subset of attributes required to identify entries within a collection (list, array, or map) of the data schema.

In the case that a path specifies an element within the data tree that does not exist, these deletes MUST be silently accepted.

3.4.7. Error Handling

When a client issues a "SetRequest", and the target is unable to apply the specified changes, an error MUST be reported to the client. The error is specified in multiple places:

- o Within a "SetResponse" message, the error field indicates the completion status of the entire transaction.

- o With a "UpdateResult" message, where the error field indicates the completion status of the individual operation.

The target MUST specify the "message" field within a "SetResponse" message such that the overall status of the transaction is reflected. In the case that no error occurs, the target MUST complete this field specifying the "OK (0)" canonical error code.

In the case that any operation within the "SetRequest" message fails, then (as per Section 3.4.3), the target MUST NOT apply any of the specified changes, and MUST consider the transaction as failed. The target SHOULD set the "message" field of the "SetResponse" message to an error message with the code field set to "Aborted (10)", and MUST set the "message" field of the "UpdateResult" corresponding to the failed operation to an "Error" message indicating failure. In the case that the processed operation is not the only operation within the "SetRequest" the target MUST set the "message" field of the "UpdateResult" messages for all other operations, setting the code field to "Aborted (10)".

For the operation that the target is unable to process, the "message" field MUST be set to a specific error code indicating the reason for failure based on the following mappings to canonical gRPC error codes:

- o When the client has specified metadata requiring authentication (see Section 3.1), and the authentication fails - "Unauthenticated (16)".

- o When the client does not have permission to modify the path specified by the operation - "PermissionDenied (7)".

- o When the operation specifies a path that cannot be parsed by the target - "InvalidArgument (3)". In this case, the "message" field of the "Error" message specified SHOULD specify human-readable text indicating that the path could not be parsed.

- When the operation is an update or replace operation that corresponds to a path that is not valid - "NotFound (5)". In this case the "message" field of the "Error" message specified SHOULD specify human-readable text indicating the path that was invalid.

- When the operation is an update or replace operation that includes an invalid value within the "Update" message specified - "InvalidArgument (3)". This error SHOULD be used in cases where the payload specifies scalar values that do not correspond to the correct schema type, and in the case that multiple values are specified using a particular encoding (e.g., JSON) which cannot be decoded by the target.

3.5. Subscribing to Telemetry Updates

When a client wishes to receive updates relating to the state of data instances on a target, it creates a subscription via the "Subscribe" RPC. A subscription consists of one or more paths, with a specified subscription mode. The mode of each subscription determines the triggers for updates for data sent from the target to the client.

All requests for new subscriptions are encapsulated within a "SubscribeRequest" message - which itself has a mode which describes the longevity of the subscription. A client may create a subscription which has a dedicated stream to return one-off data ("ONCE"); a subscription that utilizes a stream to periodically request a set of data ("POLL"); or a long-lived subscription that streams data according to the triggers specified within the individual subscription’s mode ("STREAM").

The target generates messages according to the type of subscription that has been created, at the frequency requested by the client. The methods to create subscriptions are described in Section 3.5.1.

Subscriptions are created for a set of paths - which cannot be modified throughout the lifetime of the subscription. In order to cancel a subscription, the client closes the gRPC channel over which the "Subscribe" RPC was initiated, or terminates the entire gRPC session.

Subscriptions are fundamentally a set of independent update messages relating to the state of the data tree. That is, it is not possible for a client requesting a subscription to assume that the set of update messages received represent a snapshot of the data tree at a particular point in time. Subscriptions therefore allow a client to:

- Receive ongoing updates from a target which allow synchronization between the client and target for the state of elements within the data tree. In this case (i.e., a "STREAM" subscription), a client creating a subscription receives an initial set of updates, terminated by a message indicating that initial synchronisation has completed, and then receives subsequent updates indicating changes to the initial state of those elements.

- Receive a single view (polled, or one-off) for elements of the data tree on a per-data element basis according to the state that they are in at the time that the message is transmitted. This can be more resource efficient for both target and client than a "GetRequest" for large subtrees within the data tree. The target does not need to coalesce values into a single snapshot view, or create an in-memory representation of the subtree at the time of the request, and subsequently transmit this entire view to the client.

Based on the fact that subsequent update messages are considered to be independent, and to ensure that the efficiencies described above can be achieved, by default a target MUST NOT aggregate values within an update message.

In some cases, however, elements of the data tree may be known to change together, or need to be interpreted by the subscriber together. Such data MUST be explicitly marked in the schema as being eligible to be aggregated when being published. Additionally, the subscribing client MUST explicitly request aggregation of eligible schema elements for the subscription - by means of the "allow_aggregation" flag within a "SubscriptionList" message. For elements covered by a subscription that are not explicitly marked within the schema as being eligible for aggregation the target MUST NOT coalesce these values, regardless of the value of the "allow_aggregation" flag.

When aggregation is not permitted by the client or the schema each update message MUST contain a (key, value) pair - where the key MUST be a path to a single leaf element within the data tree (encoded according to Section 2.2.2). The value MUST encode only the value of the leaf specified. In most cases, this will be a scalar value (i.e., a JSON value if a JSON encoding is utilised), but in some cases, where an individual leaf element within the schema represents an object, it MAY represent a set of values (i.e., a JSON or Protobuf object).

Where aggregation is permitted by both the client and schema, each update message MUST contain a key value pair, where the key MUST be the path to the element within the data tree which is explicitly marked as being eligible for aggregation. The value MUST be an object which encodes the children of the data tree element specified.

For JSON, the value is therefore a JSON object, and for Protobuf is a series of binary-encoded Protobuf messages.

3.5.1. Managing Subscriptions

3.5.1.1. The SubscribeRequest Message

A "SubscribeRequest" message is sent by a client to request updates from the target for a specified set of paths.

The fields of the "SubscribeRequest" are as follows:

- A group of fields, only one of which may be specified, which indicate the type of operation that the "SubscribeRequest" relates to. These are:
  - "subscribe" - a "SubscriptionList" message specifying a new set of paths that the client wishes to subscribe to.
  - "poll"- a "Poll" message used to specify (on an existing channel) that the client wishes to receive a polled update for the paths specified within the subscription. The semantics of the "Poll" message are described in Section 3.5.1.5.3.
  - "aliases" - used by a client to define (on an existing channel) a new path alias (as described in Section 2.4.2). The use of the aliases message is described in Section 3.5.1.6.

In order to create a new subscription (and its associated channel) a client MUST send a "SubscribeRequest" message, specifying the "subscribe" field. The "SubscriptionList" may create a one-off subscription, a poll-only subscription, or a streaming subscription. In the case of ONCE subscriptions, the channel between client and target MUST be closed following the initial response generation.

Subscriptions are set once, and subsequently not modified by a client. If a client wishes to subscribe to additional paths from a target, it MUST do so by sending an additional "Subscribe" RPC call, specifying a new "SubscriptionList" message. In order to end an existing subscription, a client simply closes the gRPC channel that relates to that subscription. If a channel is initiated with a "SubscribeRequest" message that does not specify a "SubscriptionList" message with the "request" field, the target MUST consider this an error. If an additional "SubscribeRequest" message specifying a "SubscriptionList" is sent via an existing channel, the target MUST respond to this message with "SubscribeResponse" message indicating an error message, with a contained error message indicating an error code of "InvalidArgument (4)"; existing subscriptions on other gRPC channels MUST not be modified or terminated.

If a client initiates a "Subscribe" RPC with a "SubscribeRequest" message which does not contain a "SubscriptionList" message, this is an error. A "SubscribeResponse" message with the contained "error" message indicating a error code of "InvalidArgument" MUST be sent. The error text SHOULD indicate that an out-of-order operation was requested on a non-existent subscription. The target MUST subsequently close the channel.

3.5.1.2. The SubscriptionList Message

A "SubscriptionList" message is used to indicate a set of paths for which common subscription behavior are required. The fields of the message are:

- "subscription" - a set of "Subscription" messages that indicate the set of paths associated with the subscription list.

- "mode" - the type of subscription that is being created. This may be "ONCE" (described in Section 3.5.1.5.1); "STREAM" (described in Section 3.5.1.5.2); or "POLL" (described in Section 3.5.1.5.3). The default value for the mode field is "STREAM".

- "prefix"- a common prefix that is applied to all paths specified within the message as per the definition in Section 2.4.1. The default prefix is null.

- "use_aliases"- a boolean flag indicating whether the client accepts target aliases via the subscription channel. In the case that such aliases are accepted, the logic described in Section 2.4.2 is utilised. By default, path aliases created by the target are not supported.

- "qos" - a field describing the packet marking that is to be utilised for the responses to the subscription that is being created. This field has a single sub-value, "marking", which indicates the DSCP value as a 32-bit unsigned integer. If the "qos" field is not specified, the device should export telemetry traffic using its default DSCP marking for management-plane traffic.

- "allow_aggregation" - a boolean value used by the client to allow schema elements that are marked as eligible for aggregation to be combined into single telemetry update messages. By default, aggregation MUST NOT be used.

- "use_models" - a "ModelData" message (as specified in Section 2.6.1) specifying the schema definition modules that the target should use when creating a subscription. When specified, the target MUST only consider data elements within the defined set of schema models as defined in Section 2.6. When "use_models" is not specified, the target MUST consider all data elements that are defined in all schema modules that it supports.

A client generating a "SubscriptionList" message MUST include the "subscription" field - which MUST be a non-empty set of "Subscription" messages, all other fields are optional.

3.5.1.3. The Subscription Message

A "Subscription" message generically describes a set of data that is to be subscribed to by a client. It contains a "path", specified as per the definition in Section 2.2.2.

There is no requirement for the path that is specified within the message to exist within the current data tree on the server. Whilst the path within the subscription SHOULD be a valid path within the set of schema modules that the target supports, subscribing to any syntactically valid path within such modules MUST be allowed. In the case that a particular path does not (yet) exist, the target MUST NOT close the channel, and instead should continue to monitor for the existence of the path, and transmit telemetry updates should it exist in the future. The target MAY send a "SubscribeResponse" message populating the error field with "NotFound (5)" to inform the client that the path does not exist at the time of subscription creation.

For "POLL" and "STREAM" subscriptions, a client may optionally specify additional parameters within the "Subscription" message. The semantics of these additional fields are described in the relevant section of this document.

3.5.1.4. The SubscribeResponse Message

A "SubscribeResponse" message is transmitted by a target to a client over an established channel created by the "Subscribe" RPC. The message contains the following fields:

- A set of fields referred to as the "response" fields, only one of which can be specified per "SubscribeResponse" message:
  - "update" - a "Notification" message providing an update value for a subscribed data entity as described in Section 3.5.2. The "update" field is also utilised when a target wishes to

create an alias within a subscription, as described in Section 3.5.2.2.

- "sync_response" - a boolean field indicating that a particular set of data values has been transmitted, used for "POLL" and "STREAM" subscriptions.

- "error" - an "Error" message transmitted to indicate an error has occurred within a particular "Subscribe" RPC call.

3.5.1.5. Creating Subscriptions

3.5.1.5.1. ONCE Subscriptions

A subscription operating in the "ONCE" mode acts as a single request/response channel. The target creates the relevant update messages, transmits them, and subsequently closes the channel.

In order to create a one-off subscription, a client sends a "SubscribeRequest" message to the target. The "subscribe" field within this message specifies a "SubscriptionList" with the mode field set to "ONCE". Updates corresponding to the subscription are generated as per the semantics described in Section 3.5.2.

Following the transmission of all updates which correspond to data items within the set of paths specified within the subscription list, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be transmitted, and the channel over which the "SubscribeRequest" was received MUST be closed.

3.5.1.5.2. STREAM Subscriptions

Stream subscriptions are long-lived subscriptions which continue to transmit updates relating to the set of paths that are covered within the subscription indefinitely.

A "STREAM" subscription is created by sending a "SubscribeRequest" message with the subscribe field containing a "SubscriptionList" message with the type specified as "STREAM". Each entry within the "Subscription" message is specified with one of the following "modes":

- On Change ("ON_CHANGE") - when a subscription is defined to be "on change", data updates are only sent when the value of the data item changes. A heartbeat interval MAY be specified along with an "on change" subscription - in this case, the value of the data item(s) MUST be re-sent once per heartbeat interval regardless of whether the value has changed or not.

- Sampled ("SAMPLE") - a subscription that is defined to be sampled MUST be specified along with a "sample_interval" encoded as an unsigned 64-bit integer representing nanoseconds. The value of the data item(s) is sent once per sample interval to the client. If the target is unable to support the desired "sample_interval" it MUST reject the subscription by returning a "SubscribeResponse" message with the error field set to an error message indicating the "InvalidArgument (3)" error code. If the "sample_interval" is set to 0, the target MUST create the subscription and send the data with the lowest interval possible for the target.

  * Optionally, the "suppress_redundant" field of the "Subscription" message may be set for a sampled subscription. In the case that it is set to "true", the target SHOULD NOT generate a telemetry update message unless the value of the path being reported on has changed since the last update was generated. Updates MUST only be generated for those individual leaf nodes in the subscription that have changed. That is to say that for a subscription to "/a/b" - where there are leaves "c" and "d" branching from the "b" node - if the value of "c" has changed, but "d" remains unchanged, an update for "d" MUST NOT be generated, whereas an update for "c" MUST be generated.

  * A "heartbeat_interval" MAY be specified to modify the behavior of "suppress_redundant" in a sampled subscription. In this case, the target MUST generate one telemetry update per heartbeat interval, regardless of whether the "suppress_redundant" flag is set to "true". This value is specified as an unsigned 64-bit integer in nanoseconds.

- Target Defined "(TARGET_DEFINED)" - when a client creates a subscription specifying the target defined mode, the target SHOULD determine the best type of subscription to be created on a per-leaf basis. That is to say, if the path specified within the message refers to some leaves which are event driven (e.g., the changing of state of an entity based on an external trigger) then an "ON_CHANGE" subscription may be created, whereas if other data represents counter values, a "SAMPLE" subscription may be created.

3.5.1.5.3. POLL Subscriptions

Polling subscriptions are used for on-demand retrieval of statistics via long-lived channels. A poll subscription relates to a certain set of subscribed paths, and is initiated by sending a "SubscribeRequest" message with encapsulated "SubscriptionList". "Subscription" messages contained within the "SubscriptionList" indicate the set of paths that are of interest to the polling client.

To retrieve data from the target, a client sends a "SubscribeRequest" message to the target, containing a "poll" field, specified to be an empty "Poll" message. On reception of such a message, the target MUST generate updates for all the corresponding paths within the "SubscriptionList". Updates MUST be generated according to Section 3.5.2.

3.5.1.6. Client-defined Aliases within a Subscription

When a client wishes to create an alias that a target should use for a path, the client should send a "SubscribeRequest" message specifying the "aliases" field. The "aliases" field consists of an "AliasList" message. An "AliasList" specifies a list of aliases, each of which consists of:

- "path" - the target path for the alias - encoded as per Section 2.2.2.

- "alias" - the (client-defined) alias for the path, encoded as per Section 2.4.2.

Where a target is unable to support a client-defined alias it SHOULD respond with a "SubscribeResponse" message with the error field indicating an error of the following types:

- "InvalidArgument (3)" where the specified alias is not acceptable to the target.

- "AlreadyExists (6)" where the alias defined is a duplicate of an existing alias for the client.

- "ResourceExhausted (8)" where the target has insufficient memory or processing resources to support the alias.

- "Unknown (2)" in all other cases.

Thus, for a client to create an alias corresponding to the path"/a/b/c/d[id=10]/e" with the name "shortPath", it sends a "SubscribeRequest" message with the following fields specified:

```
subscriberequest: <
  aliases: <
    alias: <
      path: <
        element: "a"
        element: "b"
        element: "c"
        element: "d[id=10]"
        element: "e"
      >
      alias: "#shortPath"
    >
  >
>
```

If the alias is acceptable to the target, subsequent updates are transmitted using the "#shortPath" alias in the same manner as described in Section 3.5.2.2.

3.5.2. Sending Telemetry Updates

3.5.2.1. Bundling of Telemetry Updates

Since multiple "Notification" messages can be included in the update field of a "SubscribeResponse" message, it is possible for a target to bundle messages such that fewer messages are sent to the client. The advantage of such bundling is clearly to reduce the number of bytes on the wire (caused by message overhead). Since "Notification" messages contain the timestamp at which an event occurred, or a sample was taken, such bundling does not affect the sample accuracy to the client. However, bundling does have a negative impact on the freshness of the data in the client - and on the client’s ability to react to events on the target.

Since it is not possible for the target to infer whether its clients are sensitive to the latency introduced by bundling, if a target implements optimizations such that multiple "Notification" messages are bundled together, it MUST provide an ability to disable this functionality within the configuration of the gNMI service. Additionally, a target SHOULD provide means by which the operator can control the maximum number of updates that are to be bundled into a single message, This configuration is expected to be implemented out-of-band to the gNMI protocol itself.

3.5.2.2. Target-defined Aliases within a Subscription

Where the "use_aliases" field of a "SubscriptionList" message has been set to "true", a target MAY create aliases for paths within a subscription. A target-defined alias MUST be created separately from an update to the corresponding data item(s).

To create a target-defined alias, a "SubscribeResponse" message is generated with the "update" field set to a "Notification" message. The "Notification" message specifies the aliased path within the "prefix" field, and a non-null "alias" field, specified according to Section 2.4.2.

Thus, a target wishing to create an alias relating to the path "/a/b/c[id=10]" and subsequently update children of the "c[id=10]"entity must:

1. Generate a "SubscribeResponse" message and transmit it over the channel to the client:

```
subscriberesponse: <
  update: <
    timestamp: (timestamp)
    prefix: <
      element: "a"
      element: "b"
      element: "c[id=10]"
    >
    alias: "#42"
  >
>
```

1. Subsequently, this alias can be used to provide updates for the "child1" leaf corresponding to "/a/b/c[id=10]/child1":

```
     subscriberesponse: <
       update: <
         timestamp: (timestamp)
         prefix: <
           element: "#42"
         >
         update: <
           path: <
             element: "child1"
           >
           value: <
             value: 102                      // integer representation
             type: JSON_IETF
           >
         >
       >
     >
```

3.5.2.3. Sending Telemetry Updates

When an update for a subscribed telemetry path is to be sent, a "SubscribeResponse" message is sent from the target to the client, on the channel associated with the subscription. The "update" field of the message contains a "Notification" message as per the description in Section 2.1. The "timestamp" field of the "Notification" message MUST be set to the time at which the value of the path that is being updated was collected.

Where a leaf node’s value has changed, or a new node has been created, an "Update" message specifying the path and value for the updated data item MUST be appended to the "update" field of the message.

Where a node within the subscribed paths has been removed, the "delete" field of the "Notification" message MUST have the path of the node that has been removed appended to it.

When the target has transmitted the initial updates for all paths specified within the subscription, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be transmitted to the client to indicate that the initial transmission of updates has concluded. This provides an indication to the client that all of the existing data for the subscription has been sent at least once. For "STREAM" subscriptions, such messages are not required for subsequent updates. For "POLL" subscriptions, after each set of updates for individual poll request, a "SubscribeResponse" message with the "sync_response" field set to "true" MUST be generated.

4. References

4.1. URIs

[1] http://grpc.io

[2] https://developers.google.com/protocol-buffers/

[3] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi.proto

[4] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

[5] http://www.openconfig.net/

[6] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md

[7] https://tools.ietf.org/html/rfc7159

[8] https://tools.ietf.org/html/rfc7159

[9] https://tools.ietf.org/html/rfc7951

[10] http://www.grpc.io/grpc-java/javadoc/index.html

[11] https://godoc.org/google.golang.org/grpc/codes#Code

[12] http://www.grpc.io/grpc/cpp/classgrpc_1_1_status.html

[13] https://datatracker.ietf.org/doc/draft-openconfig-netmod-model-catalog/

[14] https://tools.ietf.org/html/draft-openconfig-netmod-model-catalog-01

[15] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-authentication.md

[16] http://www.openconfig.net/documentation/semantic-versioning/

[17] https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi.proto

Appendix A. Appendix: Current Protobuf Message and Service Specification

The latest Protobuf IDL gNMI specification is found at [17].

Appendix B. Appendix: Current Outstanding Issues/Future Features

- Ability for the client to exclude paths from a subscription or get.

- "Dial out" for the target to register with an NMS and publish pre-configured subscriptions.

Authors' Addresses

Rob Shakir
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043

Email: robjs@google.com

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Paul Borman
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: borman@google.com

Marcus Hines
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: hines@google.com

Carl Lebsack
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: csl@google.com

Chris Morrow
Google

Email: christopher.morrow@gmail.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 4, 2018

Q. Wu
X. Ding
Huawei

March 3, 2018

NETCONF Base Notifications for NMDA
draft-wu-netconf-base-notification-nmda-00

Abstract

NMDA introduces additional datastores for systems that support more advanced processing chains converting configuration to operational state. Support the monitoring of the base system events pertaining to these datatores hasn’t been discussed in Network Configuration Protocol (NETCONF) Base Notifications [RFC6470]. This document updates [RFC6470] to support the Network Management Datastore Architecture (NMDA) defined in [I-D.ietf-netmod-revised-datastores].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC6470] provides standard mechanisms to support the monitoring of the base system events within the NETCONF server. Such mechanism allows a NETCONF client to receive notifications for some common system events (e.g., a change in NETCONF server capabilities, that may impact management applications.).

This document updates Network Configuration Protocol (NETCONF) Base Notifications [RFC6470] to support the Network Management Datastore Architecture (NMDA) defined in [I-D.ietf-netmod-revised-datastores]. Specifically, with NMDA, there are several additional datastores that are subject to system events. Extensions are needed to indicate the affected datastore and affected phase (because it is no longer simply about <running>; there are now also (for example) <intended> and <operational>).

The solution presented in this document is backwards compatible with [RFC6470]. This is achieved by only adding new top-level resources, and thereby leaving the semantics of all existing resources alone.

Note that "push-change-update" notification and "push-update" notification defined in [I-D.ietf-netconf-yang-push] are not general purpose notifications. and used to send to the receivers of a subscription entire or a portion of datastore contents. The solution presented in this document can work together with "push-change-update" notification and "push-update" notification defined in [I-D.ietf-netconf-yang-push] to indicate to yang push client fine granularity of data change metadata properties (e.g. who made configuration changes, identify specific location of configuration changes or phase of configuration changes) pertaining to multiple subscriptions of the same receivers.

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 [I-D.ietf-netmod-revised-datastores] and are not redefined here:

- operational state datastore
- running configuration datastore
- intended configuration datastore

2. Summary of Updates to RFC 6470

This document is intended to provide an extension of notifications initially defined within [RFC6470], with the development of NMDA architcture and data model. Key relationships between these two documents include:

- the existing notifications defined in [RFC6470] are remain unchanged, no additional information is added.
- an extra event notification is defined in this document to overcome the shortcoming of [RFC6470] for supporting NMDA.

3. NETCONF Base Notifications YANG Model extension for NMDA

3.1. Overview

The YANG module in NETCONF Base Notifications [RFC6470] specifies the following 5 event notifications for the 'NETCONF' stream to notify a client application that the NETCONF server state has changed:

- netconf-config-change
- netconf-capability-change
- netconf-session-start

- netconf-session-end

- netconf-confirmed-commit

These event notifications used within the 'NETCONF' stream are accessible to clients via the subscription mechanism described in [RFC5277].

This document extends the YANG module defined in [RFC6470] to include NMDA specific extension which allows a NETCONF client to receive notifications for additional common system event as follows:

netconf-data-change:

Generated when the NETCONF server detects that the conventional configuration datastore or 'config true' objects in the operational state datastore has been changed by a management session. The notification summarizes the edits that have been detected.

The following is an example of a netconf-data-change notification message:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-06-16T16:30:59.137045+09:00</eventTime>
 <netconf-data-change xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-notificati
ons-nmda">
        <changed-by>
            <username>admin</username>
            <session-id>0</session-id>
            <source-host>10.251.93.83</source-host>
        </changed-by>
        <datastore>operational</datastore>
        <edit>
            <target>/ietf-interfaces:interfaces/ietf-interfaces:statistics</targ
et>
            <operation>create</operation>
            <origin>default</orign>
            <current-phase>inactive</current-phase>
        </edit>
        <edit>
            <target>/ietf-interfaces:interfaces/ietf-interfaces:statistics/ietf-
interfaces:in-octets</target>
            <operation>merge</operation>
            <origin>system</orign>
            <current-phase>in-use</current-phase>
        </edit>
 </netconf-config-change>
</notification>
```

3.2. Definitions

This section presents the YANG module defined in this document.

```
 <CODE BEGINS> file "ietf-netconf-notifications-nmda@2018-02-01.yang"
module ietf-netconf-notifications-nmda {
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-notifications-nmda";
  prefix ncdn;

  import ietf-netconf {
    prefix nc;
  }
  import ietf-datastores {
    prefix ds;
  }
  import ietf-origin {
    prefix or;
  }
  import ietf-netconf-notifications {
    prefix ncn;
  }

  organization
    "IETF NETCONF (Network Configuration Protocol) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

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

     Editor:   Qin Wu
               <mailto:bill.wu@huawei.com>

     Editor:   Xiaojian Ding
               <mailto:dingxiaojian1@huawei.com>";
  description
    "This module defines a YANG data model for use with the
     NETCONF protocol that allows the NETCONF client to
     receive additional common NETCONF base event notifications
      related to NMDA.

     Copyright (c) 2012 IETF Trust and the persons identified as
     the document authors.  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 2018-02-01 {
    description
      "Initial version.";
    reference "RFC xxx: NETCONF Base Notifications for NMDA";
  }
  identity change-phase {
    description
      "Base identity for change phanse.";
  }

  identity inactive {
    base change-phase;
    description
      "Identity for inactive data.It is referred to as
      configuration that is not currently used.";
  }

  identity in-use {
    base change-phase;
    description
      "Identity for the in use data. It is referred to as
       configuration that is actively used.";
  }

  identity Remnant  {
    base in-use;
    description
      "Identity for the remnant configuration. It indicates that
       both the previous and current configuration coexist.";
  }

  identity miss-resource {
    base in-use;
    description
      "Identity for the missing resource.It indicates that
       Configuration in <intended> can refer to resources that are not
      available or otherwise not physically present and parts of <intended>
      are not applied.";
  }

  identity sys-resource {
    base in-use;
    description
      "Identity for the system congtrolled resource.It indicates that
       a system controlled resource has matching configuration in
      <intended>.";
  }
  notification netconf-data-change {
    description
      "Generated when the NETCONF server detects that the
       <operational> datastore or conventional configuration datastore
       has been changed by a management session.
       The notification summarizes the edits that
       have been detected.

       The server MAY choose to also generate this
       notification while loading a datastore during the
       boot process for the device.";
    uses ncn:changed-by-parms;
    leaf datastore {
      type identityref {
        base ds:datastore;
      }
      default "ds:operational";
      description
        "Indicates which datastore has changed or which datastore is
         target of edit-data operation.";
    }

    list edit {
      description
        "An edit record SHOULD be present for each distinct
         edit operation that the server has detected on
         the target datastore.  This list MAY be omitted
         if the detailed edit operations are not known.
         The server MAY report entries in this list for
         changes not made by a NETCONF session (e.g., CLI).";
      leaf target {
        type instance-identifier;
        description
          "Topmost node associated with the configuration change.
           A server SHOULD set this object to the node within
           the datastore that is being altered.  A server MAY
           set this object to one of the ancestors of the actual
           node that was changed, or omit this object, if the
           exact node is not known.";
      }
      leaf origin {
        type identityref {
          base or:origin;
        }
        description
          "Indicate origin that most accurately reflects the source of the
           configuration that is in use by the system.";
      }
    leaf current-phase {
        type identityref {
          base change-phase;
        }
        description
          "Indicate the current phase of the <operational> datastore or
           conventional configuration datastore change, e.g.,the intended
           datastore is validating, the intended datastore is validated,
           the configuration is applying, the configuration is applied.";
      }
      leaf operation {
        type nc:edit-operation-type;
        description
          "Type of edit operation performed.
           A server MUST set this object to the NETCONF edit
           operation performed on the target datastore.";
      }
    }
  }
}
<CODE ENDS>
```

4. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH, defined in [RFC6242].

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

/netconf-data-change:

Event type itself indicates that the system configuration has changed. This event could alert an attacker that specific configuration data nodes have been altered.

/netconf-data-change/changed-by:

Indicates whether the server or a specific user management session made the configuration change. Identifies the user name, session-id, and source host address associated with the configuration change, if any.

/netconf-data-change/datastore:

Indicates which datastore has been changed. This data can be used to determine if the running configuration data, the intended configuration data or the operational state datastore data has been changed.

/netconf-data-change/edit:

Identifies the specific edit operations and specific datastore subtree(s), specific source of configuration that have changed. the current stage of the datastore change(e.g.,inactive, in use or remnant). This data could be used to determine if specific server vulnerabilities may now be present.

5. IANA Considerations

This document registers one XML namespace URN in the ’IETF XML registry’, following the format defined in [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-notifications-nmda

Registrant Contact: The IESG.

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

This document registers one module name in the ’YANG Module Names’ registry, defined in [RFC7950]:

name: ietf-netconf-notifications-nmda

prefix: ncdn

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-notifications-nmda

RFC: xxxx

6. Acknowledgements

Thanks to Juergen Schoenwaelder and Alex Clemm to review this draft and provide important input to this document.

7. Normative References

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-15 (work in progress), February 2018.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

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

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

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

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

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <https://www.rfc-editor.org/info/rfc6021>.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

Appendix A. Appendix

A.1. Tree diagram

```
   module: ietf-netconf-notifications
   notifications:
     +---n netconf-data-change
        +--ro changed-by
        |  +--ro (server-or-user)
        |     +--:(server)
        |     |  +--ro server?        empty
        |     +--:(by-user)
        |        +--ro username       string
        |        +--ro session-id     nc:session-id-or-zero-type
        |        +--ro source-host?   inet:ip-address
        +--ro datastore?    identityref
        +--ro edit*
           +--ro target?      instance-identifier
           +--ro origin?
           +--ro current-phase?
           +--ro operation?   nc:edit-operation-type
```

Authors’ Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Xiaojian Ding
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: dingxiaojian1@huawei.com