NETCONF
Internet-Draft
Intended status: Standards Track
Expires: November 2, 2019

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
Microsoft
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
May 01, 2019

Subscription to YANG Event Notifications
draft-ietf-netconf-subscribed-notifications-25

Abstract

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher's event streams. Applying these elements allows a subscriber to request for and receive a continuous, custom feed of publisher generated information.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 2, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 3
1.1. Motivation 3
1.2. Terminology 4
1.3. Solution Overview 5
1.4. Relationship to RFC 5277 6
2. Solution 7
2.1. Event Streams 7
2.2. Event Stream Filters 8
2.3. QoS 8
2.4. Dynamic Subscriptions 9
2.5. Configured Subscriptions 17
2.6. Event Record Delivery 25
2.7. Subscription state change notifications 26
2.8. Subscription Monitoring 31
2.9. Advertisement 32
3. YANG Data Model Trees 32
3.1. Event Streams Container 32
3.2. Filters Container 33
3.3. Subscriptions Container 33
4. Data Model 35
5. Considerations 62
5.1. IANA Considerations 62
5.2. Implementation Considerations 63
5.3. Transport Requirements 64
5.4. Security Considerations 64
6. Acknowledgments 68
7. References 68

7.1. Normative References 68
7.2. Informative References 70
Appendix A. Example Configured Transport Augmentation 71
Appendix B. Changes between revisions 73
Authors' Addresses 79

1. Introduction

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher's event streams. Effectively this enables a 'subscribe then publish' capability where the customized information needs and access permissions of each target receiver are understood by the publisher before subscribed event records are marshaled and pushed. The receiver then gets a continuous, custom feed of publisher generated information.

While the functionality defined in this document is transport-agnostic, transports like NETCONF [RFC6241] or RESTCONF [RFC8040] can be used to configure or dynamically signal subscriptions, and there are bindings defined for subscribed event record delivery for NETCONF within [I-D.draft-ietf-netconf-netconf-event-notifications], and for RESTCONF within [I-D.draft-ietf-netconf-restconf-notif].

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

1.1. Motivation

Various limitations in [RFC5277] are discussed in [RFC7923]. Resolving these issues is the primary motivation for this work. Key capabilities supported by this document include:

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

1.2. Terminology

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

Client: defined in [RFC8342].

Configuration: defined in [RFC8342].

Configuration datastore: defined in [RFC8342].

Configured subscription: A subscription installed via configuration into a configuration datastore.

Dynamic subscription: A subscription created dynamically by a subscriber via a remote procedure call.

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

Event occurrence time: a timestamp matching the time an originating process identified as when an event happened.

Event record: A set of information detailing an event.

Event stream: A continuous, chronologically ordered set of events aggregated under some context.

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

Notification message: Information intended for a receiver indicating that one or more events have occurred.

Publisher: An entity responsible for streaming notification messages per the terms of a subscription.

Receiver: A target to which a publisher pushes subscribed event records. For dynamic subscriptions, the receiver and subscriber are the same entity.

Subscriber: A client able to request and negotiate a contract for the generation and push of event records from a publisher. For dynamic subscriptions, the receiver and subscriber are the same entity.

Subscription: A contract with a publisher, stipulating which information one or more receivers wish to have pushed from the publisher without the need for further solicitation.

All YANG tree diagrams used in this document follow the notation defined in [RFC8340].

1.3. Solution Overview

This document describes a transport agnostic mechanism for subscribing to and receiving content from an event stream within a publisher. This mechanism is through the use of a subscription.

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via a Remote Procedure Call (RPC). If the publisher is able to serve this request, it accepts it, and then starts pushing notification messages back to the subscriber. If the publisher is not able to serve it as requested, then an error response is returned. This response MAY include hints at subscription parameters that, had they been present, may have enabled the dynamic subscription request to be accepted.

2. Configured subscriptions, which allow the management of subscriptions via a configuration so that a publisher can send notification messages to a receiver. Support for configured subscriptions is optional, with its availability advertised via a YANG feature.

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bound by the transport session used to establish it. For connection-oriented stateful transports like NETCONF, the loss of the transport session will result in the immediate termination of any associated dynamic subscriptions. For connectionless or stateless transports like HTTP, a lack of receipt acknowledgment of a sequential set of notification messages and/or keep-alives can be used to trigger a termination of a dynamic subscription. Contrast this to the lifetime of a configured subscription. This lifetime is driven by relevant configuration being present within the publisher's

applied configuration. Being tied to configuration operations implies configured subscriptions can be configured to persist across reboots, and implies a configured subscription can persist even when its publisher is fully disconnected from any network.

- Configured subscriptions can be modified by any configuration client with write permission on the configuration of the subscription. Dynamic subscriptions can only be modified via an RPC request made by the original subscriber, or a change to configuration data referenced by the subscription.

Note that there is no mixing-and-matching of dynamic and configured operations on a single subscription. Specifically, a configured subscription cannot be modified or deleted using RPCs defined in this document. Similarly, a dynamic subscription cannot be directly modified or deleted by configuration operations. It is however possible to perform a configuration operation which indirectly impacts a dynamic subscription. By changing value of a pre-configured filter referenced by an existing dynamic subscription, the selected event records passed to a receiver might change.

Also note that transport specific transport specifications based on this specification MUST detail the lifecycle of dynamic subscriptions, as well as the lifecycle of configured subscriptions (if supported).

A publisher MAY terminate a dynamic subscription at any time. Similarly, it MAY decide to temporarily suspend the sending of notification messages for any dynamic subscription, or for one or more receivers of a configured subscription. Such termination or suspension is driven by internal considerations of the publisher.

1.4. Relationship to RFC 5277

This document is intended to provide a superset of the subscription capabilities initially defined within [RFC5277]. Especially when extending an existing [RFC5277] implementation, it is important to understand what has been reused and what has been replaced. Key relationships between these two documents include:

- this document defines a transport independent capability, [RFC5277] is specific to NETCONF.

- the data model in this document is used instead of the data model in Section 3.4 of [RFC5277] for the new operations.

- the RPC operations in this draft replace the operation "create-subscription" defined in [RFC5277], section 4.

- the <notification> message of [RFC5277], Section 4 is used.
- the included contents of the "NETCONF" event stream are identical between this document and [RFC5277].
- a publisher MAY implement both the Notification Management Schema and RPCs defined in [RFC5277] and this new document concurrently.
- unlike [RFC5277], this document enables a single transport session to intermix notification messages and RPCs for different subscriptions.
- A subscription "stop-time" can be specified as part of a notification replay. This supports an analogous capability to the stopTime parameter of [RFC5277]. However in this specification, a "stop-time" parameter can also be applied without replay.

2. Solution

Per the overview provided in Section 1.3, this section details the overall context, state machines, and subsystems which may be assembled to allow the subscription of events from a publisher.

2.1. Event Streams

An event stream is a named entity on a publisher which exposes a continuously updating set of YANG defined event records. An event record is an instantiation of a "notification" YANG statement. If the "notification" is defined as a child to a data node, the instantiation includes the hierarchy of nodes that identifies the data node in the datastore (see Section 7.16.2 of [RFC7950]). Each event stream is available for subscription. It is out of the scope of this document to identify a) how event streams are defined (other than the NETCONF stream), b) how event records are defined/generated, and c) how event records are assigned to event streams.

There is only one reserved event stream name within this document: "NETCONF". The "NETCONF" event stream contains all NETCONF event record information supported by the publisher, except where an event record has explicitly been excluded from the stream. Beyond the "NETCONF" stream, implementations MAY define additional event streams.

As YANG defined event records are created by a system, they may be assigned to one or more streams. The event record is distributed to a subscription's receiver(s) where: (1) a subscription includes the identified stream, and (2) subscription filtering does not exclude the event record from that receiver.

Access control permissions may be used to silently exclude event records from within an event stream for which the receiver has no read access. As an example of how this might be accomplished, see [RFC8341] section 3.4.6. Note that per Section 2.7 of this document, subscription state change notifications are never filtered out.

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

Event records MUST NOT be delivered to a receiver in a different order than they were placed onto an event stream.

2.2. Event Stream Filters

This document defines an extensible filtering mechanism. The filter itself is a boolean test which is placed on the content of an event record. A 'false' filtering result causes the event record to be excluded from delivery to a receiver. A filter never results in information being stripped from within an event record prior to that event record being encapsulated within a notification message. The two optional event stream filtering syntaxes supported are [XPATH] and subtree [RFC6241].

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

2.3. QoS

This document provides for several Quality of Service (QoS) parameters. These parameters indicate the treatment of a subscription relative to other traffic between publisher and receiver. Included are:

- A "dscp" marking to differentiate prioritization of notification messages during network transit.

- A "weighting" so that bandwidth proportional to this weighting can be allocated to this subscription relative to other subscriptions.

- a "dependency" upon another subscription.

If the publisher supports the "dscp" feature, then a subscription with a "dscp" leaf MUST result in a corresponding [RFC2474] DSCP marking being placed within the IP header of any resulting notification messages and subscription state change notifications.

Where TCP is used, a publisher which supports the "dscp" feature SHOULD ensure that a subscription's notification messages are returned within a single TCP transport session where all traffic shares the subscription's "dscp" leaf value. Where this cannot be guaranteed, any "establish subscription" RPC request SHOULD be rejected with a "dscp-unavailable" error.

For the "weighting" parameter, when concurrently dequeuing notification messages from multiple subscriptions to a receiver, the publisher MUST allocate bandwidth to each subscription proportionally to the weights assigned to those subscriptions. "Weighting" is an optional capability of the publisher; support for it is identified via the "qos" feature.

If a subscription has the "dependency" parameter set, then any buffered notification messages containing event records selected by the parent subscription MUST be dequeued prior to the notification messages of the dependent subscription. If notification messages have dependencies on each other, the notification message queued the longest MUST go first. If a "dependency" included within an RPC references a subscription which does not exist or is no longer accessible to that subscriber, that "dependency" MUST be silently removed. "Dependency" is an optional capability of the publisher; support for it is identified via the "qos" feature.

2.4. Dynamic Subscriptions

Dynamic subscriptions are managed via protocol operations (in the form of [RFC7950], Section 7.14 RPCs) made against targets located within the publisher. These RPCs have been designed extensibly so that they may be augmented for subscription targets beyond event streams. For examples of such augmentations, see the RPC augmentations within [I-D.ietf-netconf-yang-push]'s YANG model.

2.4.1. Dynamic Subscription State Model

Below is the publisher's state machine for a dynamic subscription. Each state is shown in its own box. It is important to note that such a subscription doesn't exist at the publisher until an "establish-subscription" RPC is accepted. The mere request by a subscriber to establish a subscription is insufficient for that subscription to be externally visible. Start and end states are depicted to reflect subscription creation and deletion events.

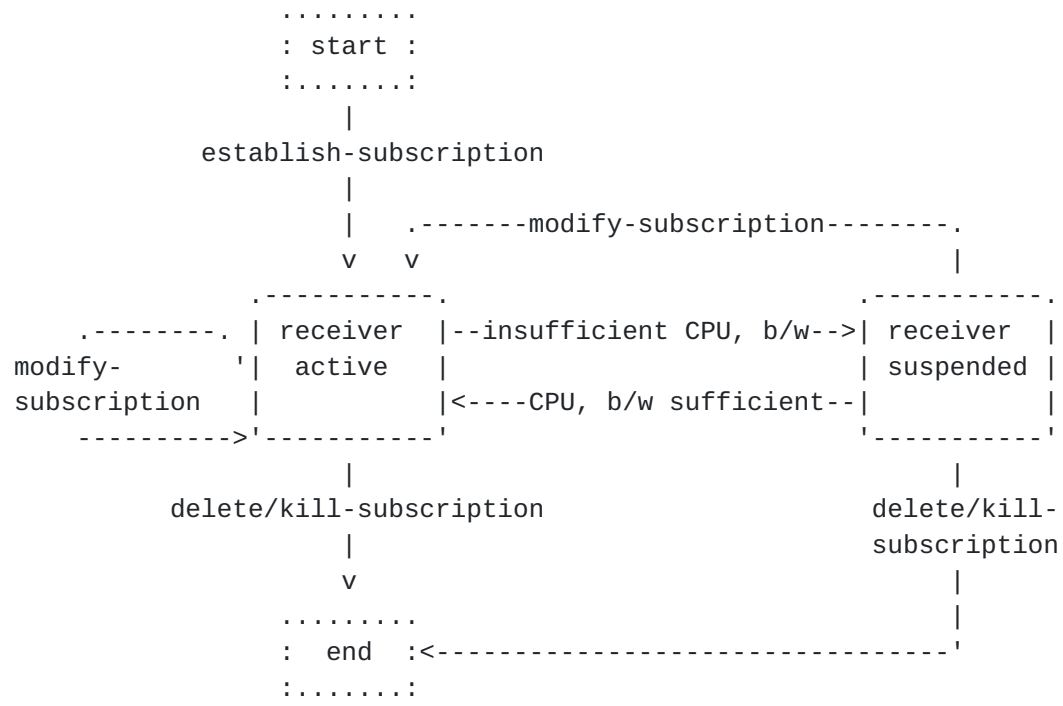


Figure 1: Publisher's state for a dynamic subscription

Of interest in this state machine are the following:

- Successful "establish-subscription" or "modify-subscription" RPCs put the subscription into the active state.

- Failed "modify-subscription" RPCs will leave the subscription in its previous state, with no visible change to any streaming updates.

- A "delete-subscription" or "kill-subscription" RPC will end the subscription, as will the reaching of a "stop-time".

- A publisher may choose to suspend a subscription when there is insufficient CPU or bandwidth available to service the subscription. This is notified to a subscriber with a "subscription-suspended" subscription state change notification.

- A suspended subscription may be modified by the subscriber (for example in an attempt to use fewer resources). Successful modification returns the subscription to the active state.

- Even without a "modify-subscription" request, a publisher may return a subscription to the active state should the resource constraints become sufficient again. This is announced to the

subscriber via the "subscription-resumed" subscription state change notification.

2.4.2. Establishing a Dynamic Subscription

The "establish-subscription" RPC allows a subscriber to request the creation of a subscription.

The input parameters of the operation are:

- A "stream" name which identifies the targeted event stream against which the subscription is applied.
- An event stream filter which may reduce the set of event records pushed.
- Where the transport used by the RPC supports multiple encodings, an optional "encoding" for the event records pushed. 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 "replay-start-time" for the subscription. The "replay-start-time" MUST be in the past and indicates that the subscription is requesting a replay of previously generated information from the event stream. For more on replay, see Section 2.4.2.1. Where there is no "replay-start-time", the subscription starts immediately.

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

Below is a tree diagram for "establish-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
      +---x establish-subscription
         +---w input
         |  +---w (target)
         |  |  +--:(stream)
         |  |     +---w (stream-filter)?
         |  |     |  +--:(by-reference)
         |  |     |  |  +---w stream-filter-name
         |  |     |  |          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}?
         |  +---w stop-time?
         |  |       yang:date-and-time
         |  +---w dscp?                                       inet:dscp
         |  |       {dscp}?
         |  +---w weighting?                                  uint8
         |  |       {qos}?
         |  +---w dependency?
         |  |       subscription-id {qos}?
         |  +---w encoding?                                   encoding
         +--ro output
            +--ro id                             subscription-id
            +--ro replay-start-time-revision?    yang:date-and-time
                    {replay}?
```

Figure 2: establish-subscription RPC tree diagram

A publisher MAY reject the "establish-subscription" RPC for many reasons as described in Section 2.4.6. The contents of the resulting RPC error response MAY include details on input parameters which if considered in a subsequent "establish-subscription" RPC, may result in a successful subscription establishment. Any such hints MUST be transported within a yang-data "establish-subscription-stream-error-info" container included within the RPC error response.

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

Figure 3: establish-subscription RPC yang-data tree diagram

2.4.2.1. Requesting a replay of event records

Replay provides the ability to establish a subscription which is also capable of passing event records generated in the recent past. In other words, as the subscription initializes itself, it sends any event records within the target event stream which meet the filter criteria, which have an event time which is after the "replay-start-time", and which have an event time before the "stop-time" should this "stop-time" exist. The end of these historical event records is identified via a "replay-completed" subscription state change notification. Any event records generated since the subscription establishment may then follow. For a particular subscription, all event records will be delivered in the order they are placed into the event stream.

Replay is an optional feature which is dependent on an event stream supporting some form of logging. This document puts no restrictions on the size or form of the log, where it resides within the publisher, or when event record entries in the log are purged.

The inclusion of a "replay-start-time" within an "establish-subscription" RPC indicates a replay request. If the "replay-start-time" contains a value that is earlier than what a publisher's retained history supports, then if the subscription is accepted, the actual publisher's revised start time MUST be set in the returned "replay-start-time-revision" object.

A "stop-time" parameter may be included in a replay subscription. For a replay subscription, the "stop-time" MAY be earlier than the current time, but MUST be later than the "replay-start-time".

If the given "replay-start-time" is later than the time marked within any event records retained within the replay buffer, then the publisher MUST send a "replay-completed" notification immediately after a successful establish-subscription RPC response.

If an event stream supports replay, the "replay-support" leaf is present in the "/streams/stream" list entry for the event stream. An event stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given replay request. To assess the timeframe available for replay,

subscribers can read the leafs "replay-log-creation-time" and "replay-log-aged-time". See Figure 18 for the YANG tree, and Section 4 for the YANG model describing these elements. The actual size of the replay log at any given time is a publisher specific matter. Control parameters for the replay log are outside the scope of this document.

2.4.3. Modifying a Dynamic Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription. Dynamic subscriptions can be modified any number of times. Dynamic subscriptions can only be modified via this RPC using a transport session connecting to the subscriber. If the publisher accepts the requested modifications, it acknowledges success to the subscriber, then immediately starts sending event records based on the new terms.

Subscriptions created by configuration cannot be modified via this RPC. However configuration may be used to modify objects referenced by the subscription (such as a referenced filter).

Below is a tree diagram for "modify-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
    +---x modify-subscription
       +---w input
          +---w id
          |       subscription-id
          +---w (target)
          |  +--:(stream)
          |     +---w (stream-filter)?
          |        +--:(by-reference)
          |        |  +---w stream-filter-name
          |        |          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 stop-time?
                  yang:date-and-time
```

Figure 4: modify-subscription RPC tree diagram

If the publisher accepts the requested modifications on a currently suspended subscription, the subscription will immediately be resumed (i.e., the modified subscription is returned to the active state.) The publisher MAY immediately suspend this newly modified subscription through the "subscription-suspended" notification before any event records are sent.

If the publisher rejects the RPC request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. Rejection of the RPC for any reason is indicated by via RPC error as described in Section 2.4.6. The contents of such a rejected RPC MAY include hints on inputs which (if considered) may result in a successfully modified subscription. These hints MUST be transported within a yang-data "modify-subscription-stream-error-info" container inserted into the RPC error response.

Below is a tree diagram for "modify-subscription-RPC-yang-data". All objects contained in this tree are described within the included YANG model within Section 4.

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

Figure 5: modify-subscription RPC yang-data tree diagram

2.4.4. Deleting a Dynamic Subscription

The "delete-subscription" operation permits canceling an existing subscription. If the publisher accepts the request, and the publisher has indicated success, the publisher MUST NOT send any more notification messages for this subscription.

Below is a tree diagram for "delete-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

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

Figure 6: delete-subscription RPC tree diagram

Dynamic subscriptions can only be deleted via this RPC using a transport session connecting to the subscriber. Configured subscriptions cannot be deleted using RPCs.

2.4.5. Killing a Dynamic Subscription

The "kill-subscription" operation permits an operator to end a dynamic subscription which is not associated with the transport session used for the RPC. A publisher MUST terminate any dynamic subscription identified by the "id" parameter in the RPC request, if such a subscription exists.

Configured subscriptions cannot be killed using this RPC. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to kill a configured subscription.

Below is a tree diagram for "kill-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x kill-subscription
  +---w input
     +---w id     subscription-id
```

Figure 7: kill-subscription RPC tree diagram

2.4.6. RPC Failures

Whenever an RPC is unsuccessful, the publisher returns relevant information as part of the RPC error response. Transport level error processing MUST be done before RPC error processing described in this section. In all cases, RPC error information returned will use existing transport layer RPC structures, such as those seen with NETCONF in [RFC6241] Appendix A, or with RESTCONF in [RFC8040] Section 7.1. These structures MUST be able to encode subscription specific errors identified below and defined within this document's YANG model.

As a result of this variety, how subscription errors are encoded within an RPC error response is transport dependent. Following are valid errors which can occur for each RPC:

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

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

To see a NETCONF based example of an error response from above, see [I-D.draft-ietf-netconf-netconf-event-notifications], Figure 10.

There is one final set of transport independent RPC error elements included in the YANG model. These are three yang-data structures which enable the publisher to provide to the receiver that error information which does not fit into existing transport layer RPC structures. These three yang-data structures are:

1. "establish-subscription-stream-error-info": This MUST be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

2. "modify-subscription-stream-error-info": This MUST be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere within the transport portion of a failed "modify-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

3. "delete-subscription-error-info": This MUST be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

2.5. Configured Subscriptions

A configured subscription is a subscription installed via configuration. Configured subscriptions may be modified by any configuration client with the proper permissions. Subscriptions can be modified or terminated via configuration at any point of their lifetime. Multiple configured subscriptions MUST be supportable over a single transport session.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions:

- persistence across publisher reboots,
- persistence even when transport is unavailable, and
- an ability to send notification messages to more than one receiver (note that receivers are unaware of the existence of any other receivers.)

On the publisher, supporting configured subscriptions is optional and advertised using the "configured" feature. On a receiver of a configured subscription, support for dynamic subscriptions is optional. However if replaying missed event records is required for a configured subscription, support for dynamic subscription is highly recommended. In this case, a separate dynamic subscription can be established to retransmit the missing event records.

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

- A "transport" which identifies the transport protocol to use to connect with all subscription receivers.
- One or more receivers, each intended as the destination for event records. Note that each individual receiver is identifiable by its "name".
- Optional parameters to identify where traffic should egress a publisher:
  - A "source-interface" which identifies the egress interface to use from the publisher. Publisher support for this is optional and advertised using the "interface-designation" feature.
  - A "source-address" address, which identifies the IP address to stamp on notification messages destined for the receiver.
  - A "source-vrf" which identifies the VRF on which to reach receivers. This VRF is a network instance as defined within [I-D.draft-ietf-rtgwg-ni-model]. Publisher support for VRFs is optional and advertised using the "supports-vrf" feature.

  If none of the above parameters are set, notification messages MUST egress the publisher's default interface.

A tree diagram describing these parameters is shown in Figure 20 within Section 3.3. All parameters are described within the YANG model in Section 4.

2.5.1. Configured Subscription State Model

Below is the state machine for a configured subscription on the publisher. This state machine describes the three states (valid, invalid, and concluded), as well as the transitions between these states. Start and end states are depicted to reflect configured subscription creation and deletion events. The creation or modification of a configured subscription initiates an evaluation by the publisher to determine if the subscription is in valid or invalid states. The publisher uses its own criteria in making this determination. If in the valid state, the subscription becomes operational. See (1) in the diagram below.

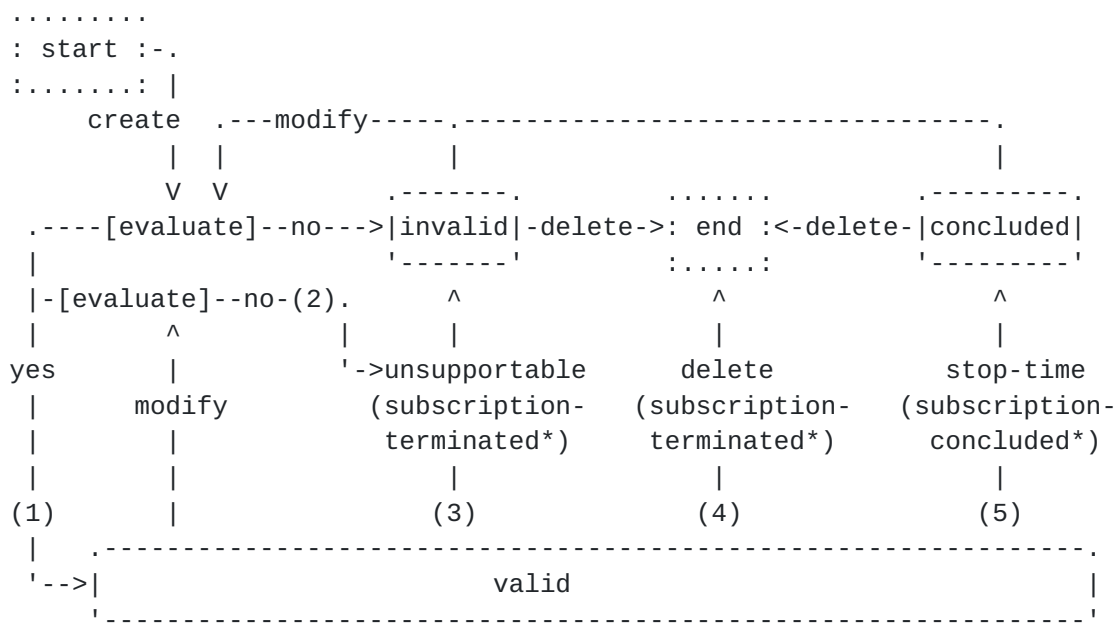
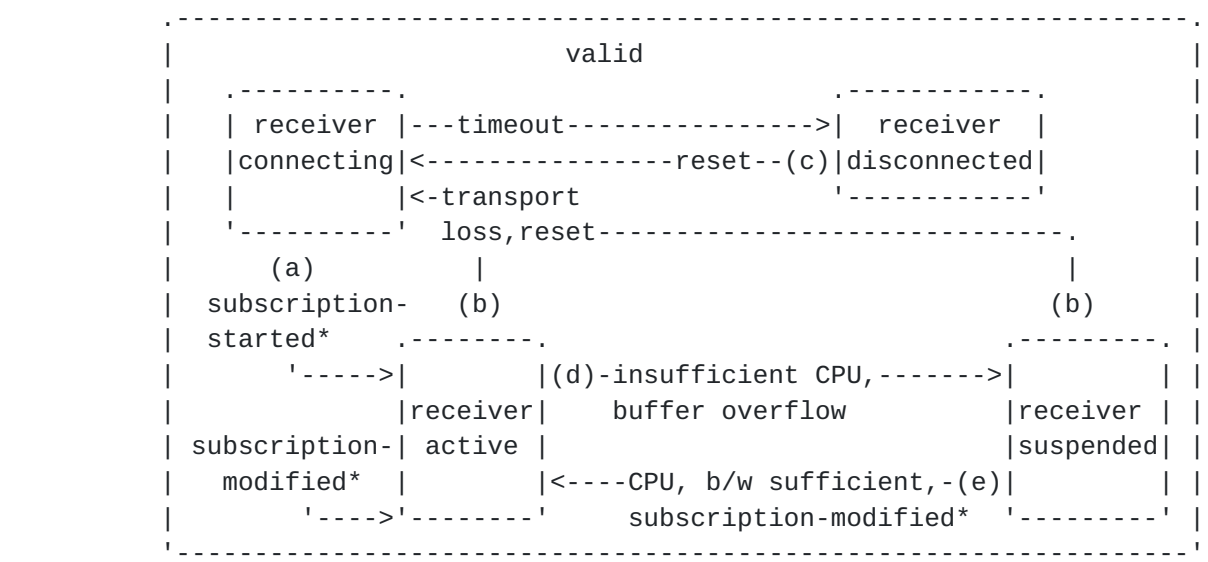


Figure 8: Publisher state model for a configured subscription

A subscription in the valid state may move to the invalid state in one of two ways. First, it may be modified in a way which fails a re-evaluation. See (2) in the diagram. Second, the publisher might determine that the subscription is no longer supportable. This could

be for reasons of an unexpected but sustained increase in an event stream's event records, degraded CPU capacity, a more complex referenced filter, or other higher priority subscriptions which have usurped resources. See (3) in the diagram. No matter the case, a "subscription-terminated" notification is sent to any receivers in an active or suspended state. A subscription in the valid state may also transition to the concluded state via (5) if a configured stop time has been reached. In this case, a "subscription-concluded" notification is sent to any receivers in active or suspended states. Finally, a subscription may be deleted by configuration (4).

When a subscription is in the valid state, a publisher will attempt to connect with all receivers of a configured subscription and deliver notification messages. Below is the state machine for each receiver of a configured subscription. This receiver state machine is fully contained within the state machine of the configured subscription, and is only relevant when the configured subscription is in the valid state.



Legend:
dashed boxes which include the word 'receiver' show the possible states for an individual receiver of a valid configured subscription.
* indicates a subscription state change notification

Figure 9: Receiver state for a configured subscription on a Publisher

When a configured subscription first moves to the valid state, the "state" leaf of each receiver is initialized to the connecting state. If transport connectivity is not available to any receiver and there are any notification messages to deliver, a transport session is established (e.g., through [RFC8071]). Individual receivers are

moved to the active state when a "subscription-started" subscription state change notification is successfully passed to that receiver (a). Event records are only sent to active receivers. Receivers of a configured subscription remain active if both transport connectivity can be verified to the receiver, and event records are not being dropped due to a publisher buffer capacity being reached. The result is that a receiver will remain active on the publisher as long as events aren't being lost, or the receiver cannot be reached. In addition, a configured subscription's receiver MUST be moved to the connecting state if the receiver is reset via the "reset" action (b), (c). For more on reset, see Section 2.5.5. If transport connectivity cannot be achieved while in the connecting state, the receiver MAY be moved to the disconnected state.

A configured subscription's receiver MUST be moved to the suspended state if there is transport connectivity between the publisher and receiver, but notification messages are failing to be delivered due to publisher buffer capacity being reached, or notification messages are not able to be generated for that receiver due to insufficient CPU (d). This is indicated to the receiver by the "subscription-suspended" subscription state change notification.

A configured subscription receiver MUST be returned to the active state from the suspended state when notification messages are able to be generated, bandwidth is sufficient to handle the notification messages, and a receiver has successfully been sent a "subscription-resumed" or "subscription-modified" subscription state change notification (e). The choice as to which of these two subscription state change notifications is sent is determined by whether the subscription was modified during the period of suspension.

Modification of a configured subscription is possible at any time. A "subscription-modified" subscription state change notification will be sent to all active receivers, immediately followed by notification messages conforming to the new parameters. Suspended receivers will also be informed of the modification. However this notification will await the end of the suspension for that receiver (e).

The mechanisms described above are mirrored in the RPCs and notifications within the document. It should be noted that these RPCs and notifications have been designed to be extensible and allow subscriptions into targets other than event streams. For instance, the YANG module defined in Section 5 of [I-D.ietf-netconf-yang-push] augments "/sn:modify-subscription/sn:input/sn:target".

2.5.2. Creating a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level "subscriptions" subtree.

Because there is no explicit association with an existing transport session, configuration operations MUST include additional parameters beyond those of dynamic subscriptions. These parameters identify each receiver, how to connect with that receiver, and possibly whether the notification messages need to come from a specific egress interface on the publisher. Receiver specific transport connectivity parameters MUST be configured via transport specific augmentations to this specification. See Section 2.5.7 for details.

After a subscription is successfully established, the publisher immediately sends a "subscription-started" subscription 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 capacity being reached, a new "subscription-started" must be sent. This new "subscription-started" indicates an event record discontinuity.

To see an example of subscription creation using configuration operations over NETCONF, see Appendix A of [I-D.draft-ietf-netconf-netconf-event-notifications].

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 subscription state change notification "subscription-terminated" is sent to that receiver if that receiver is active or suspended.

If the modification involves changing the policies for the subscription, the publisher sends to currently active receivers a "subscription-modified" notification. For any suspended receivers, a

"subscription-modified" notification will be delayed until the receiver is resumed. (Note: in this case, the "subscription-modified" notification informs the receiver that the subscription has been resumed, so no additional "subscription-resumed" need be sent. Also note that if multiple modifications have occurred during the suspension, only the "subscription-modified" notification describing the latest one need be sent to the receiver.)

2.5.4. Deleting a Configured Subscription

Subscriptions can be deleted through configuration against the top-level "subscriptions" subtree.

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription a subscription state change notification stating the subscription has ended (i.e., "subscription-terminated").

2.5.5. Resetting a Configured Subscription Receiver

It is possible that a configured subscription to a receiver needs to be reset. This is accomplished via the "reset" action within the YANG model at "/subscriptions/subscription/receivers/receiver/reset". This action may be useful in cases where a publisher has timed out trying to reach a receiver. When such a reset occurs, a transport session will be initiated if necessary, and a new "subscription-started" notification will be sent. This action does not have any effect on transport connectivity if the needed connectivity already exists.

2.5.6. Replay for a Configured Subscription

It is possible to do replay on a configured subscription. This is supported via the configuration of the "configured-replay" object on the subscription. The setting of this object enables the streaming of the buffered event records for the subscribed event stream. All buffered event records which have been retained since the last publisher restart will be sent to each configured receiver.

Replay of events records created since restart is useful. It allows event records generated before transport connectivity establishment to be passed to a receiver. Setting the restart time as the earliest configured replay time precludes possibility of resending of event records logged prior to publisher restart. It also ensures the same records will be sent to each configured receiver, regardless of the speed of transport connectivity establishment to each receiver. Finally, establishing restart as the earliest potential time for

event records to be included within notification messages, a well-understood timeframe for replay is defined.

As a result, when any configured subscription receivers become active, buffered event records will be sent immediately after the "subscription-started" notification. If the publisher knows the last event record sent to a receiver, and the publisher has not rebooted, the next event record on the event stream which meets filtering criteria will be the leading event record sent. Otherwise, the leading event record will be the first event record meeting filtering criteria subsequent to the latest of three different times: the "replay-log-creation-time", "replay-log-aged-time", or the most recent publisher boot time. The "replay-log-creation-time" and "replay-log-aged-time" are discussed in Section 2.4.2.1. The most recent publisher boot time ensures that duplicate event records are not replayed from a previous time the publisher was booted.

It is quite possible that a receiver might want to retrieve event records from an event stream prior to the latest boot. If such records exist where there is a configured replay, the publisher MUST send the time of the event record immediately preceding the "replay-start-time" within the "replay-previous-event-time" leaf. Through the existence of the "replay-previous-event-time", the receiver will know that earlier events prior to reboot exist. In addition, if the subscriber was previously receiving event records with the same subscription "id", the receiver can determine if there was a time gap where records generated on the publisher were not successfully received. And with this information, the receiver may choose to dynamically subscribe to retrieve any event records placed into the event stream before the most recent boot time.

All other replay functionality remains the same as with dynamic subscriptions as described in Section 2.4.2.1.

2.5.7. Transport Connectivity for a Configured Subscription

This specification is transport independent. However supporting a configured subscription will often require the establishment of transport connectivity. And the parameters used for this transport connectivity establishment are transport specific. As a result, the YANG model defined within Section 4 is not able to directly define and expose these transport parameters.

It is necessary for an implementation to support the connection establishment process. To support this function, the YANG model does include a node where transport specific parameters for a particular receiver may be augmented. This node is "/subscriptions/subscription/receivers/receiver". By augmenting

transport parameters from this node, system developers are able to incorporate the YANG objects necessary to support the transport connectivity establishment process.

The result of this is the following requirement. A publisher supporting the feature "configured" MUST also support least one YANG model which augments transport connectivity parameters on "/subscriptions/subscription/receivers/receiver". For an example of such an augmentation, see Appendix A.

2.6. Event Record Delivery

Whether dynamic or configured, once a subscription has been set up, the publisher streams event records via notification messages per the terms of the subscription. For dynamic subscriptions, notification messages are sent over the session used to establish the subscription. For configured subscriptions, notification messages are sent over the connections specified by the transport and each receiver of a configured subscription.

A notification message is sent to a receiver when an event record is not blocked by either the specified filter criteria or receiver permissions. This notification message MUST include an "eventTime" object as defined per [RFC5277] Section 4. This "eventTime" MUST be at the top level of YANG structured event record.

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 10: subscribed notification message

When a dynamic subscription has been started or modified, with "establish-subscription" or "modify-subscription" respectively, event records matching the newly applied filter criteria MUST NOT be sent until after the RPC reply has been sent.

When a configured subscription has been started or modified, event records matching the newly applied filter criteria MUST NOT be sent

until after the "subscription-started" or "subscription-modified" notifications has been sent, respectively.

2.7. Subscription state change notifications

In addition to sending event records to receivers, a publisher MUST also send subscription state change notifications when events related to subscription management have occurred.

Subscription state change notifications are unlike other notifications in that they are never included in any event stream. Instead, they are inserted (as defined in this section) within the sequence of notification messages sent to a particular receiver. subscription state change notifications cannot be dropped or filtered out, they cannot be stored in replay buffers, and they are delivered only to impacted receivers of a subscription. The identification of subscription state change notifications is easy to separate from other notification messages through the use of the YANG extension "subscription-state-notif". This extension tags a notification as a subscription state change notification.

The complete set of subscription state change 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 subscription state change notification are all the parameters of the subscription, except for the receiver(s) transport connection information and origin information indicating where notification messages will egress the publisher. Note that if a referenced filter from the "filters" container has been used within the subscription, the notification still provides the contents of that referenced filter under the "within-subscription" subtree.

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

Below is a tree diagram for "subscription-started". All objects contained in this tree are described within the included YANG model within Section 4.

```
      +---n subscription-started {configured}?
         +--ro id
         |       subscription-id
         +--ro (target)
         |  +--:(stream)
         |     +--ro (stream-filter)?
         |     |  +--:(by-reference)
         |     |  |  +--ro stream-filter-name
         |     |  |          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}?
         |     +--ro replay-previous-event-time?
         |             yang:date-and-time {replay}?
         +--ro stop-time?
         |       yang:date-and-time
         +--ro dscp?                                      inet:dscp
         |       {dscp}?
         +--ro weighting?                                 uint8 {qos}?
         +--ro dependency?
         |       subscription-id {qos}?
         +--ro transport?                                 transport
         |       {configured}?
         +--ro encoding?                                  encoding
         +--ro purpose?                                   string
                 {configured}?
```

Figure 11: subscription-started notification tree diagram

2.7.2. subscription-modified

This notification indicates that a subscription has been modified by configuration operations. It is delivered directly after the last event records processed using the previous subscription parameters, and before any event records processed after the modification.

Below is a tree diagram for "subscription-modified". All objects contained in this tree are described within the included YANG model within Section 4.

```
      +---n subscription-modified
         +--ro id
         |       subscription-id
         +--ro (target)
         |  +--:(stream)
         |     +--ro (stream-filter)?
         |     |  +--:(by-reference)
         |     |  |  +--ro stream-filter-name
         |     |  |          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}?
         +--ro stop-time?
         |       yang:date-and-time
         +--ro dscp?                                      inet:dscp
         |       {dscp}?
         +--ro weighting?                                 uint8 {qos}?
         +--ro dependency?
         |       subscription-id {qos}?
         +--ro transport?                                 transport
         |       {configured}?
         +--ro encoding?                                  encoding
         +--ro purpose?                                   string
                 {configured}?
```

Figure 12: subscription-modified notification tree diagram

A publisher most often sends this notification directly after the modification of any configuration parameters impacting a configured subscription. But it may also be sent at two other times:

1. Where a configured subscription has been modified during the suspension of a receiver, the notification will be delayed until the receiver's suspension is lifted. In this situation, the notification indicates that the subscription has been both modified and resumed.

2. A "subscription-modified" subscription state change notification MUST be sent if the contents of the filter identified by the subscription's "stream-filter-ref" leaf has changed. This state

change notification is to be sent for a filter change impacting any active receiver of a configured or dynamic subscription.

2.7.3. subscription-terminated

This notification indicates that no further event records for this subscription should be expected from the publisher. A publisher may terminate the sending event records to a receiver for the following reasons:

1. Configuration which removes a configured subscription, or a "kill-subscription" RPC which ends a dynamic subscription. These are identified via the reason "no-such-subscription".

2. A referenced filter is no longer accessible. This is identified by "filter-unavailable".

3. The event stream referenced by a subscription is no longer accessible by the receiver. This is identified by "stream-unavailable".

4. A suspended subscription has exceeded some timeout. This is identified by "suspension-timeout".

Each of the reasons above correspond one-to-one with a "reason" identityref specified within the YANG model.

Below is a tree diagram for "subscription-terminated". All objects contained in this tree are described within the included YANG model within Section 4.

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

Figure 13: subscription-terminated notification tree diagram

Note: this subscription state change notification MUST be sent to a dynamic subscription's receiver when the subscription ends unexpectedly. The cases when this might happen are when a "kill-subscription" RPC is successful, or when some other event not including the reaching the subscription's "stop-time" results in a publisher choosing to end the subscription.

2.7.4. subscription-suspended

This notification indicates that a publisher has suspended the sending of event records to a receiver, and also indicates the possible loss of events. Suspension happens when capacity constraints stop a publisher from serving a valid subscription. The two conditions where is this possible are:

1. "insufficient-resources" when a publisher is unable to produce the requested event stream of notification messages, and

2. "unsupportable-volume" when the bandwidth needed to get generated notification messages to a receiver exceeds a threshold.

These conditions are encoded within the "reason" object. No further notification will be sent until the subscription resumes or is terminated.

Below is a tree diagram for "subscription-suspended". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-suspended
   +--ro id        subscription-id
   +--ro reason    identityref
```

Figure 14: subscription-suspended notification tree diagram

2.7.5. subscription-resumed

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

Below is the tree diagram for "subscription-resumed". All objects contained in this tree are described within the included YANG model within Section 4.

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

Figure 15: subscription-resumed notification tree diagram

2.7.6. subscription-completed

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

Below is a tree diagram for "subscription-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-completed {configured}?
   +--ro id    subscription-id
```

Figure 16: subscription-completed notification tree diagram

2.7.7. replay-completed

This notification indicates that all of the event records prior to the current time have been passed to a receiver. It is sent before any notification message containing an event record with a timestamp later than (1) the "stop-time" or (2) the subscription's start time.

If a subscription contains no "stop-time", or has a "stop-time" that has not been reached, then after the "replay-completed" notification has been sent, additional event records will be sent in sequence as they arise naturally on the publisher.

Below is a tree diagram for "replay-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n replay-completed {replay}?
   +--ro id    subscription-id
```

Figure 17: replay-completed notification tree diagram

2.8. Subscription Monitoring

In the operational state datastore, the container "subscriptions" maintains the state of all dynamic subscriptions, as well as all configured subscriptions. Using datastore retrieval operations, or subscribing to the "subscriptions" container [I-D.ietf-netconf-yang-push] allows the state of subscriptions and their connectivity to receivers to be monitored.

Each subscription in the operational state datastore is represented as a list element. Included in this list are event counters for each receiver, the state of each receiver, as well as the subscription parameters currently in effect. The appearance of the leaf "configured-subscription-state" indicates that a particular subscription came into being via configuration. This leaf also indicates if the current state of that subscription is valid, invalid, and concluded.

To understand the flow of event records within a subscription, there are two counters available for each receiver. The first counter is "sent-event-records" which shows the quantity of events actually identified for sending to a receiver. The second counter is "excluded-event-records" which shows event records not sent to receiver. "excluded-event-records" shows the combined results of both access control and per-subscription filtering. For configured subscriptions, counters are reset whenever the subscription is evaluated to valid (see (1) in Figure 8).

Dynamic subscriptions are removed from the operational state datastore once they expire (reaching stop-time) or when they are terminated. While many subscription objects are shown as configurable, dynamic subscriptions are only included within the operational state datastore and as a result are not configurable.

2.9. Advertisement

Publishers supporting this document MUST indicate support of the YANG model "ietf-subscribed-notifications" within the YANG library of the publisher. In addition if supported, the optional features "encode-xml", "encode-json", "configured" "supports-vrf", "qos", "xpath", "subtree", "interface-designation", "dscp", and "replay" MUST be indicated.

3. YANG Data Model Trees

This section contains tree diagrams for nodes defined in Section 4. For tree diagrams of subscription state change notifications, see Section 2.7. For the tree diagrams for the RPCs, see Section 2.4.

3.1. Event Streams Container

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. This enables subscribers to discover what streams a publisher supports.

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

Figure 18: Stream Container tree diagram

Above is a tree diagram for the "streams" container. All objects contained in this tree are described within the included YANG model within Section 4.

3.2. Filters Container

The "filters" container maintains a list of all subscription filters that persist outside the life-cycle of a single subscription. This enables pre-defined filters which may be referenced by more than one subscription.

```
  +--rw filters
     +--rw stream-filter* [name]
        +--rw name                              string
        +--rw (filter-spec)?
           +--:(stream-subtree-filter)
           |  +--rw stream-subtree-filter?      <anydata> {subtree}?
           +--:(stream-xpath-filter)
              +--rw stream-xpath-filter?        yang:xpath1.0 {xpath}?
```

Figure 19: Filter Container tree diagram

Above is a tree diagram for the filters container. All objects contained in this tree are described within the included YANG model within Section 4.

3.3. Subscriptions Container

The "subscriptions" container maintains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which a publisher is serving.

```
  +--rw subscriptions
```

```
        +--rw subscription* [id]
           +--rw id
           |       subscription-id
           +--rw (target)
           |  +--:(stream)
           |     +--rw (stream-filter)?
           |     |  +--:(by-reference)
           |     |  |  +--rw stream-filter-name
           |     |  |          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
           |     +--ro replay-start-time?
           |     |       yang:date-and-time {replay}?
           |     +--rw configured-replay?                    empty
           |             {configured,replay}?
           +--rw stop-time?
           |       yang:date-and-time
           +--rw dscp?                                       inet:dscp
           |       {dscp}?
           +--rw weighting?                                  uint8 {qos}?
           +--rw dependency?
           |       subscription-id {qos}?
           +--rw transport?                                  transport
           |       {configured}?
           +--rw encoding?                                   encoding
           +--rw purpose?                                    string
           |       {configured}?
           +--rw (notification-message-origin)? {configured}?
           |  +--:(interface-originated)
           |  |  +--rw source-interface?
           |  |          if:interface-ref {interface-designation}?
           |  +--:(address-originated)
           |     +--rw source-vrf?
           |     |       -> /ni:network-instances/network-instance/name
           |     |       {supports-vrf}?
           |     +--rw source-address?
           |             inet:ip-address-no-zone
           +--ro configured-subscription-state?              enumeration
           |       {configured}?
           +--rw receivers
              +--rw receiver* [name]
```

```
                +--rw name                       string
                +--ro sent-event-records?
                |       yang:zero-based-counter64
                +--ro excluded-event-records?
                |       yang:zero-based-counter64
                +--ro state                      enumeration
                +---x reset {configured}?
                   +--ro output
                      +--ro time    yang:date-and-time
```

Figure 20: Subscriptions tree diagram

Above is a tree diagram for the subscriptions container. All objects contained in this tree are described within the included YANG model within Section 4.

4. Data Model

This module imports typedefs from [RFC6991], [RFC8343], and [RFC8040], and it references [I-D.draft-ietf-rtgwg-ni-model], [XPATH], [RFC6241], [RFC7049], [RFC7540], [RFC7951] , [RFC7950] and [RFC8259].

[note to the RFC Editor - please replace XXXX within this YANG model with the number of this document, and XXXY with the number of [I-D.draft-ietf-rtgwg-ni-model]]

[note to the RFC Editor - please replace the two dates within the YANG module with the date of publication]

```
<CODE BEGINS> file "ietf-subscribed-notifications@2019-05-01.yang"
module ietf-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
```

```
   }
   import ietf-netconf-acm {
     prefix nacm;
     reference
       "RFC 8341: Network Configuration Access Control Model";
   }
   import ietf-network-instance {
     prefix ni;
     reference
       "draft-ietf-rtgwg-ni-model-12: YANG Model for Network Instances";
   }
   import ietf-restconf   {
     prefix rc;
     reference
       "RFC 8040: RESTCONF Protocol";
   }
   import ietf-yang-types {
     prefix yang;
     reference
       "RFC 6991: Common YANG Data Types";
   }

   organization "IETF NETCONF (Network Configuration) Working Group";
   contact
     "WG Web:   <http:/tools.ietf.org/wg/netconf/>
      WG List:  <mailto:netconf@ietf.org>

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

      Author:   Eric Voit
                <mailto:evoit@cisco.com>

      Author:   Alberto Gonzalez Prieto
                <mailto:alberto.gonzalez@microsoft.com>

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

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

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

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

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

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

   revision 2019-05-01 {
     description
       "Initial version";
     reference
     "RFC XXXX:Subscription to YANG Event Notifications";
   }

   /*
    * FEATURES
    */

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

   feature dscp {
     description
       "This feature indicates that a publisher supports the ability to
       set the DiffServ Code Point (DSCP) value in outgoing packets.";
   }

   feature encode-json {
     description
       "This feature indicates that JSON encoding of notification
        messages is supported.";
   }

   feature encode-xml {
     description
       "This feature indicates that XML encoding of notification
        messages is supported.";
   }

   feature interface-designation {
     description
       "This feature indicates a publisher supports sourcing all
       receiver interactions for a configured subscription from a single
```

```
      designated egress interface.";
  }

  feature qos {
    description
      "This feature indicates a publisher supports absolute
      dependencies of one subscription's traffic over another, as well
      as weighted bandwidth sharing between subscriptions.  Both of
      these are Quality of Service (QoS) features which allow
      differentiated treatment of notification messages between a
      publisher and a specific receiver.";
  }

  feature replay {
    description
      "This feature indicates that historical event record replay is
      supported.  With replay, it is possible for past event records to
      be streamed in chronological order.";
  }

  feature subtree {
    description
      "This feature indicates support for YANG subtree filtering.";
    reference "RFC 6241, Section 6.";
  }

  feature supports-vrf {
    description
      "This feature indicates a publisher supports VRF configuration
      for configured subscriptions.  VRF support for dynamic
      subscriptions does not require this feature.";
    reference "RFC XXXY, Section 6.";
  }

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

  /*
   * EXTENSIONS
   */

  extension subscription-state-notification {
    description
      "This statement applies only to notifications. It indicates that
       the notification is a subscription state change notification.
```

```
        Therefore it does not participate in a regular event stream and
        does not need to be specifically subscribed to in order to be
        received. This statement can only occur as a substatement to the
        YANG 'notification' statement.  This statement is not for use
        outside of this YANG module.";
    }

    /*
     * IDENTITIES
     */

    /* Identities for RPC and Notification errors */

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

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

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

    identity dscp-unavailable {
      base establish-subscription-error;
      if-feature "dscp";
      description
```

```
       "The publisher is unable mark notification messages with a
       prioritization information in a way which will be respected
       during network transit.";
   }

   identity encoding-unsupported {
     base establish-subscription-error;
     description
       "Unable to encode notification messages in the desired format.";
   }

   identity filter-unavailable {
     base subscription-terminated-reason;
     description
      "Referenced filter does not exist.  This means a receiver is
      referencing a filter which doesn't exist, or to which they do not
      have access permissions.";
   }

   identity filter-unsupported {
     base establish-subscription-error;
     base modify-subscription-error;
     description
      "Cannot parse syntax within the filter.  This failure can be from
      a syntax error, or a syntax too complex to be processed by the
      publisher.";
   }

   identity insufficient-resources {
     base establish-subscription-error;
     base modify-subscription-error;
     base subscription-suspended-reason;
     description
       "The publisher has insufficient resources to support the
        requested subscription.  An example might be that allocated CPU
        is too limited to generate the desired set of notification
        messages.";
   }

   identity no-such-subscription {
     base modify-subscription-error;
     base delete-subscription-error;
     base subscription-terminated-reason;
     description
      "Referenced subscription doesn't exist. This may be as a result of
       a non-existent subscription id, an id which belongs to another
       subscriber, or an id for configured subscription.";
   }
```

```
   identity replay-unsupported {
     base establish-subscription-error;
     if-feature "replay";
     description
      "Replay cannot be performed for this subscription. This means the
       publisher will not provide the requested historic information
       from the event stream via replay to this receiver.";
   }

   identity stream-unavailable {
     base subscription-terminated-reason;
     description
      "Not a subscribable event stream.  This means the referenced event
       stream is not available for subscription by the receiver.";
   }

   identity suspension-timeout {
     base subscription-terminated-reason;
     description
      "Termination of previously suspended subscription. The publisher
       has eliminated the subscription as it exceeded a time limit for
       suspension.";
   }

   identity unsupportable-volume {
     base subscription-suspended-reason;
     description
       "The publisher does not have the network bandwidth needed to get
       the volume of generated information intended for a receiver.";
   }

   /* Identities for encodings */

   identity configurable-encoding {
     description
       "If a transport identity derives from this identity, it means
        that it supports configurable encodings.  An example of a
        configurable encoding might be a new identity such as
        'encode-cbor'.  Such an identity could use
        'configurable-encoding' as its base.  This would allow a
        dynamic subscription encoded in JSON [RFC-8259] to request
        notification messages be encoded via CBOR [RFC-7049].  Further
        details for any specific configurable encoding would be
        explored in a transport document based on this specification.";
   }

   identity encoding {
     description
```

```
      "Base identity to represent data encodings";
  }

  identity encode-xml {
    base encoding;
    if-feature "encode-xml";
    description
      "Encode data using XML as described in RFC 7950";
    reference
      "RFC 7950 - The YANG 1.1 Data Modeling Language";
  }

  identity encode-json {
    base encoding;
    if-feature "encode-json";
    description
      "Encode data using JSON as described in RFC 7951";
    reference
      "RFC 7951 - JSON Encoding of Data Modeled with YANG";
  }

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

  /*
   * TYPEDEFs
   */

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

  typedef stream-filter-ref {
    type leafref {
      path "/sn:filters/sn:stream-filter/sn:name";
    }
    description
      "This type is used to reference an event stream filter.";
  }
```

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

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

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

   /*
    * GROUPINGS
    */

   grouping stream-filter-elements {
     description
       "This grouping defines the base for filters applied to event
        streams.";
     choice filter-spec {
       description
         "The content filter specification for this request.";
       anydata stream-subtree-filter {
         if-feature "subtree";
         description
           "Event stream evaluation criteria encoded in the syntax of a
           subtree filter as defined in RFC 6241, Section 6.

           The subtree filter is applied to the representation of
           individual, delineated event records as contained within the
           event stream.

           If the subtree filter returns a non-empty node set, the
           filter matches the event record, and the event record is
           included in the notification message sent to the receivers.";
         reference "RFC 6241, Section 6.";
```

```
      }
      leaf stream-xpath-filter {
        if-feature "xpath";
        type yang:xpath1.0;
        description
          "Event stream evaluation criteria encoded in the syntax of
           an XPath 1.0 expression.

           The XPath expression is evaluated on the representation of
           individual, delineated event records as contained within
           the event stream.

           The result of the XPath expression is converted to a
           boolean value using the standard XPath 1.0 rules.  If the
           boolean value is 'true', the filter matches the event
           record, and the event record is included in the notification
           message sent to the receivers.

           The expression is evaluated in the following XPath context:

             o   The set of namespace declarations is the set of prefix
                 and namespace pairs for all YANG modules implemented
                 by the server, where the prefix is the YANG module
                 name and the namespace is as defined by the
                 'namespace' statement in the YANG module.

                 If the leaf is encoded in XML, all namespace
                 declarations in scope on the 'stream-xpath-filter'
                 leaf element are added to the set of namespace
                 declarations.  If a prefix found in the XML is
                 already present in the set of namespace declarations,
                 the namespace in the XML is used.

             o  The set of variable bindings is empty.

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

             o  The context node is the root node.";
        reference
          "http://www.w3.org/TR/1999/REC-xpath-19991116
           RFC 7950, Section 10.";

      }
    }
  }

  grouping update-qos {
```

```
    description
      "This grouping describes Quality of Service information
       concerning a subscription.  This information is passed to lower
       layers for transport prioritization and treatment";
    leaf dscp {
      if-feature "dscp";
      type inet:dscp;
      default "0";
      description
        "The desired network transport priority level. This is the
         priority set on notification messages encapsulating the
         results of the subscription.  This transport priority is
         shared for all receivers of a given subscription.";
    }
    leaf weighting {
      if-feature "qos";
      type uint8 {
         range "0 .. 255";
      }
      description
        "Relative weighting for a subscription. Larger weights get
         more resources. Allows an underlying transport layer perform
         informed load balance allocations between various
         subscriptions";
      reference
        "RFC-7540, section 5.3.2";
    }
    leaf dependency {
      if-feature "qos";
      type subscription-id;
      description
        "Provides the 'subscription-id' of a parent subscription which
         has absolute precedence should that parent have push updates
         ready to egress the publisher. In other words, there should be
         no streaming of objects from the current subscription if
         the parent has something ready to push.

         If a dependency is asserted via configuration or via RPC, but
         the referenced 'subscription-id' does not exist, the
         dependency is silently discarded.  If a referenced
         subscription is deleted this dependency is removed.";
      reference
        "RFC-7540, section 5.3.1";
    }
  }

  grouping subscription-policy-modifiable {
    description
```

```
      "This grouping describes all objects which may be changed
      in a subscription.";
    choice target {
      mandatory true;
      description
        "Identifies the source of information against which a
        subscription is being applied, as well as specifics on the
        subset of information desired from that source.";
      case stream {
        choice stream-filter {
          description
            "An event stream filter can be applied to a subscription.
            That filter will come either referenced from a global list,
            or be provided within the subscription itself.";
          case by-reference {
            description
              "Apply a filter that has been configured separately.";
            leaf stream-filter-name {
              type stream-filter-ref;
              mandatory true;
              description
                "References an existing event stream filter which is to
                be applied to an event stream for the subscription.";
            }
          }
          case within-subscription {
            description
              "Local definition allows a filter to have the same
              lifecycle as the subscription.";
            uses stream-filter-elements;
          }
        }
      }
    }
    leaf stop-time {
      type yang:date-and-time;
      description
        "Identifies a time after which notification messages for a
        subscription should not be sent.  If 'stop-time' is not
        present, the notification messages will continue until the
        subscription is terminated.  If 'replay-start-time' exists,
        'stop-time' must be for a subsequent time. If
        'replay-start-time' doesn't exist, 'stop-time' when established
        must be for a future time.";
    }
  }

  grouping subscription-policy-dynamic {
```

```
    description
      "This grouping describes the only information concerning a
       subscription which can be passed over the RPCs defined in this
       model.";
    uses subscription-policy-modifiable {
      augment target/stream {
        description
          "Adds additional objects which can be modified by RPC.";
        leaf stream {
          type stream-ref {
            require-instance false;
          }
          mandatory true;
          description
            "Indicates the event stream to be considered for
            this subscription.";
        }
        leaf replay-start-time {
          if-feature "replay";
          type yang:date-and-time;
          config false;
          description
            "Used to trigger the replay feature for a dynamic
            subscription, with event records being selected needing to
            be at or after the start at the time specified.  If
            'replay-start-time' is not present, this is not a replay
            subscription and event record push should start
            immediately. It is never valid to specify start times that
            are later than or equal to the current time.";
        }
      }
    }
    uses update-qos;
  }

  grouping subscription-policy {
    description
      "This grouping describes the full set of policy information
      concerning both dynamic and configured subscriptions, with the
      exclusion of both receivers and networking information specific
      to the publisher such as what interface should be used to
      transmit notification messages.";
    uses subscription-policy-dynamic;
    leaf transport {
      if-feature "configured";
      type transport;
      description
        "For a configured subscription, this leaf specifies the
```

```
        transport used to deliver messages destined to all receivers
        of that subscription.";
    }
    leaf encoding {
      when 'not(../transport) or derived-from(../transport,
      "sn:configurable-encoding")';
      type encoding;
      description
        "The type of encoding for notification messages.   For a
        dynamic subscription, if not included as part of an establish-
        subscription RPC, the encoding will be populated with the
        encoding used by that RPC.  For a configured subscription, if
        not explicitly configured the encoding with be the default
        encoding for an underlying transport.";
    }
    leaf purpose {
      if-feature "configured";
      type string;
      description
        "Open text allowing a configuring entity to embed the
        originator or other specifics of this subscription.";
    }
  }

  /*
   * RPCs
   */

  rpc establish-subscription {
    description
      "This RPC allows a subscriber to create (and possibly negotiate)
       a subscription on its own behalf.  If successful, the
       subscription remains in effect for the duration of the
       subscriber's association with the publisher, or until the
       subscription is terminated. In case an error occurs, or the
       publisher cannot meet the terms of a subscription, an RPC error
       is returned, the subscription is not created.  In that case, the
       RPC reply's 'error-info' MAY include suggested parameter
       settings that would have a higher likelihood of succeeding in a
       subsequent 'establish-subscription' request.";
    input {
      uses subscription-policy-dynamic;
      leaf encoding {
        type encoding;
        description
          "The type of encoding for the subscribed data. If not
          included as part of the RPC, the encoding MUST be set by the
          publisher to be the encoding used by this RPC.";
```

```
      }
    }
    output {
      leaf id {
        type subscription-id;
        mandatory true;
        description
          "Identifier used for this subscription.";
      }
      leaf replay-start-time-revision {
        if-feature "replay";
        type yang:date-and-time;
          description
            "If a replay has been requested, this represents the
            earliest time covered by the event buffer for the requested
            event stream.  The value of this object is the
            'replay-log-aged-time' if it exists.  Otherwise it is the
            'replay-log-creation-time'.  All buffered event records
            after this time will be replayed to a receiver.  This
            object will only be sent if the starting time has been
            revised to be later than the time requested by the
            subscriber.";
      }
    }
  }

  rc:yang-data establish-subscription-stream-error-info {
    container establish-subscription-stream-error-info {
      description
        "If any 'establish-subscription' RPC parameters are
        unsupportable against the event stream, a subscription is not
        created and the RPC error response MUST indicate the reason
        why the subscription failed to be created. This yang-data MAY
        be inserted as structured data within a subscription's RPC
        error response to indicate the failure reason.  This yang-data
        MUST be inserted if hints are to be provided back to the
        subscriber.";
      leaf reason {
        type identityref {
          base establish-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be created to a targeted event stream.";
        }
      leaf filter-failure-hint {
        type string;
          description
```

```
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription. The syntax and
             semantics of this hint are implementation-specific.";
      }
    }
  }

  rpc modify-subscription {
    description
      "This RPC allows a subscriber to modify a dynamic subscription's
       parameters.  If successful, the changed subscription
       parameters remain in effect for the duration of the
       subscription, until the subscription is again modified, or until
       the subscription is terminated.  In case of an error or an
       inability to meet the modified parameters, the subscription is
       not modified and the original subscription parameters remain in
       effect. In that case, the RPC error MAY include 'error-info'
       suggested parameter hints that would have a high likelihood of
       succeeding in a subsequent 'modify-subscription' request.  A
       successful 'modify-subscription' will return a suspended
       subscription to an 'active' state.";
    input {
      leaf id {
        type subscription-id;
        mandatory true;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-modifiable;
    }
  }

  rc:yang-data modify-subscription-stream-error-info {
    container modify-subscription-stream-error-info {
      description
        "This yang-data MAY be provided as part of a subscription's RPC
        error response when there is a failure of a
        'modify-subscription' RPC which has been made against an event
        stream.  This yang-data MUST be used if hints are to be
        provided back to the subscriber.";
      leaf reason {
        type identityref {
          base modify-subscription-error;
        }
        description
          "Information in a 'modify-subscription' RPC error response
          which indicates the reason why the subscription to an event
          stream has failed to be modified.";
```

```
      }
      leaf filter-failure-hint {
        type string;
          description
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription.";
      }
    }
  }

  rpc delete-subscription {
    description
      "This RPC allows a subscriber to delete a subscription that
       was previously created from by that same subscriber using the
       'establish-subscription' RPC.

       If an error occurs, the server replies with an 'rpc-error' where
       the 'error-info' field MAY contain an
       'delete-subscription-error-info' structure.";
    input {
      leaf id {
        type subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted.
           Only subscriptions that were created using
           'establish-subscription' from the same origin as this RPC
           can be deleted via this RPC.";
      }
    }
  }

  rpc kill-subscription {
    nacm:default-deny-all;
    description
      "This RPC allows an operator to delete a dynamic subscription
       without restrictions on the originating subscriber or underlying
       transport session.

       If an error occurs, the server replies with an 'rpc-error' where
       the 'error-info' field MAY contain an
       'delete-subscription-error-info' structure.";
    input {
      leaf id {
        type subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted. Only
```

```
            subscriptions that were created using
            'establish-subscription' can be deleted via this RPC.";
        }
      }
    }

    rc:yang-data delete-subscription-error-info {
      container delete-subscription-error-info {
        description
          "If a 'delete-subscription' RPC or a 'kill-subscription' RPC
          fails, the subscription is not deleted and the RPC error
          response MUST indicate the reason for this failure. This
          yang-data MAY be inserted as structured data within a
          subscription's RPC error response to indicate the failure
          reason.";
        leaf reason {
          type identityref {
            base delete-subscription-error;
          }
          mandatory true;
          description
            "Indicates the reason why the subscription has failed to be
            deleted.";
        }
      }
    }

    /*
     * 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.";
      leaf id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
    }

    notification subscription-completed {
      sn:subscription-state-notification;
      if-feature "configured";
```

```
    description
      "This notification is sent to indicate that a subscription has
       finished passing event records, as the 'stop-time' has been
       reached.";
    leaf id {
      type subscription-id;
      mandatory true;
      description
        "This references the gracefully completed subscription.";
    }
  }

  notification subscription-modified {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription has been
       modified.  Notification messages sent from this point on will
       conform to the modified terms of the subscription.  For
       completeness, this subscription state change notification
       includes both modified and non-modified aspects of a
       subscription.";
    leaf id {
      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-name' 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 id {
      type subscription-id;
```

```
      mandatory true;
      description
        "This references the affected 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.";
    leaf id {
      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 is using for the streaming
           of buffered event records.  This will be populated with the
           most recent of the following: the event time of the previous
           event record sent to a receiver, the
           'replay-log-creation-time', the 'replay-log-aged-time',
           or the most recent publisher boot time.";
      }
      refine "target/stream/stream-filter/within-subscription" {
        description
          "Filter applied to the subscription.  If the
          'stream-filter-name' is populated, the filter within the
          subscription came from the 'filters' container.  Otherwise it
          is populated in-line as part of the subscription.";
      }
      augment "target/stream" {
        description
          "This augmentation adds additional parameters specific to a
          subscription-started notification.";
        leaf replay-previous-event-time {
          when "../replay-start-time";
          if-feature "replay";
          type yang:date-and-time;
            description
            "If there is at least one event in the replay buffer prior
            to 'replay-start-time', this gives the time of the event
            generated immediately prior to the 'replay-start-time'.
```

```
            If a receiver previously received event records for this
            configured subscription, it can compare this time to the
            last event record previously received.  If the two are not
            the same (perhaps due to a reboot), then a dynamic replay
            can be initiated to acquire any missing event records.";
        }
      }
    }
  }

  notification subscription-suspended {
    sn:subscription-state-notification;
    description
      "This notification indicates that a suspension of the
       subscription by the publisher has occurred.  No further
       notifications will be sent until the subscription resumes.
       This notification shall only be sent to receivers of a
       subscription; it does not constitute a general-purpose
       notification.";
    leaf id {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    leaf reason {
      type identityref {
        base subscription-suspended-reason;
      }
      mandatory true;
      description
        "Identifies the condition which resulted in the suspension.";
    }
  }

  notification subscription-terminated {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription has been
       terminated.";
    leaf id {
      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 .";
    }
  }


  /*
   * DATA NODES
   */

  container streams {
    config false;
    description
      "This container contains information on the built-in event
      streams provided by the publisher.";
    list stream {
      key "name";
      description
        "Identifies the built-in event streams that are supported by
         the publisher.";
      leaf name {
        type string;
        description
          "A handle for a system-provided event stream made up of a
          sequential set of event records, each of which is
          characterized by its own domain and semantics.";
      }
      leaf description {
        type string;
        description
          "A description of the event stream, including such
           information as the type of event records that are available
           within this event stream.";
      }
      leaf replay-support {
        if-feature "replay";
        type empty;
        description
          "Indicates that event record replay is available on this
          event stream.";
      }
      leaf replay-log-creation-time {
        when "../replay-support";
        if-feature "replay";
        type yang:date-and-time;
```

```
        mandatory true;
        description
          "The timestamp of the creation of the log used to support the
          replay function on this event stream. This time might be
          earlier than the earliest available information contained in
          the log. This object is updated if the log resets for some
          reason.";
      }
      leaf replay-log-aged-time {
        when "../replay-support";
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp associated with last event record which has
           been aged out of the log.  This timestamp identifies how far
           back into history this replay log extends, if it doesn't
           extend back to the 'replay-log-creation-time'.  This object
           MUST be present if replay is supported and any event records
           have been aged out of the log.";
      }
    }
  }

  container filters {
    description
      "This container contains a list of configurable filters
       that can be applied to subscriptions.  This facilitates
       the reuse of complex filters once defined.";
    list stream-filter {
      key "name";
      description
        "A list of pre-configured filters that can be applied to
        subscriptions.";
      leaf name {
        type string;
        description
          "An name 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 "id";
        description
          "The identity and specific parameters of a subscription.
           Subscriptions within this list can be created using a control
           channel or RPC, or be established through configuration.

           If configuration operations or the 'kill-subscription' RPC are
           used to delete a subscription, a 'subscription-terminated'
           message is sent to any active or suspended receivers.";
        leaf id {
          type subscription-id;
          description
            "Identifier of a subscription; unique within a publisher";
        }
        uses subscription-policy {
          refine "target/stream/stream" {
            description
              "Indicates the event stream to be considered for this
              subscription.  If an event stream has been removed,
              and no longer can be referenced by an active subscription,
              send a 'subscription-terminated' notification with
              'stream-unavailable' as the reason.  If a configured
              subscription refers to a non-existent event stream, move
              that subscription to the 'invalid' state.";
          }
          refine "transport" {
            description
              "For a configured subscription, this leaf specifies the
              transport used to deliver messages destined to all
              receivers of that subscription.  This object is mandatory
              for subscriptions in the configuration datastore.  This
              object is not mandatory for dynamic subscriptions within
              the operational state datastore.  The object should not
              be present for dynamic subscriptions.";
          }
          augment "target/stream" {
            description
              "Enables objects to added to a configured stream
              subscription";
            leaf configured-replay {
              if-feature "configured";
              if-feature "replay";
              type empty;
              description
                "The presence of this leaf indicates that replay for the
                configured subscription should start at the earliest time
```

```
            in the event log, or at the publisher boot time, which
            ever is later.";
        }
      }
    }
    choice notification-message-origin {
      if-feature "configured";
      description
        "Identifies the egress interface on the publisher from which
         notification messages are to be sent.";
      case interface-originated {
        description
          "When notification messages to egress a specific,
           designated interface on the publisher.";
        leaf source-interface {
          if-feature "interface-designation";
          type if:interface-ref;
          description
            "References the interface for notification messages.";
        }
      }
      case address-originated {
        description
          "When notification messages are to depart from a publisher
           using specific originating address and/or routing context
           information.";
        leaf source-vrf {
          if-feature "supports-vrf";
          type leafref {
            path "/ni:network-instances/ni:network-instance/ni:name";
          }
          description
            "VRF from which notification messages should egress a
            publisher.";
        }
        leaf source-address {
          type inet:ip-address-no-zone;
          description
            "The source address for the notification messages.  If a
            source VRF exists, but this object doesn't, a publisher's
            default address for that VRF must be used.";
        }
      }
    }
    leaf configured-subscription-state {
      if-feature "configured";
      type enumeration {
        enum valid {
```

```
            value 1;
            description
              "Subscription is supportable with current parameters.";
          }
          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,
                it no longer has receivers in the 'receiver active' or
                'receiver suspended' state, but not yet been
                removed from configuration.";
          }
        }
        config false;
        description
          "The presence of this leaf indicates that the subscription
          originated from configuration, not through a control channel
          or RPC.  The value indicates the system established state
          of the subscription.";
      }
      container receivers {
        description
          "Set of receivers in a subscription.";
        list receiver {
          key "name";
          min-elements 1;
          description
            "A host intended as a recipient for the notification
            messages of a subscription.  For configured subscriptions,
            transport specific network parameters (or a leafref to
            those parameters) may augmentated to a specific receiver
            within this list.";
          leaf name {
            type string;
            description
              "Identifies a unique receiver for a subscription.";
          }
          leaf sent-event-records {
            type yang:zero-based-counter64;
            config false;
            description
              "The number of event records sent to the receiver.  The
```

```
            count is initialized when a dynamic subscription is
            established, or when a configured receiver
            transitions to the valid state.";
          }
          leaf excluded-event-records {
            type yang:zero-based-counter64;
            config false;
            description
              "The number of event records explicitly removed either
              via an event stream filter or an access control filter so
              that they are not passed to a receiver.  This count is
              set to zero each time 'sent-event-records' is
              initialized.";
          }
          leaf state {
            type enumeration {
              enum active {
                value 1;
                description
                  "Receiver is currently being sent any applicable
                  notification messages for the subscription.";
              }
              enum suspended {
                value 2;
                description
                  "Receiver 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' subscription state change
                  notification needs to be successfully received before
                  notification messages are sent.

                  If the 'reset' action is invoked for a receiver of an
                  active configured subscription, the state must be
                  moved to 'connecting'.";
              }
              enum disconnected {
                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 publisher is
              currently generating notification messages intended for
              that receiver.";
          }
          action reset {
            if-feature "configured";
            description
              "Allows the reset of this configured subscription
              receiver to the 'connecting' state. This enables the
              connection process to be re-initiated.";
            output {
              leaf time {
                type yang:date-and-time;
                mandatory true;
                description
                  "Time a publisher returned the receiver to a
                  'connecting' state.";
              }
            }
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

5. Considerations

5.1. 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-11.txt (RFC form)

5.2. Implementation Considerations

To support deployments including both configured and dynamic subscriptions, it is recommended to split the subscription "id" domain into static and dynamic halves. That way it eliminates the possibility of collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated. A best practice is to use lower half the "id" object's integer space when that "id" is assigned by an external entity (such as with a configured subscription). This leaves the upper half of subscription integer space available to be dynamically assigned by the publisher.

If a subscription is unable to marshal a series of filtered event records into transmittable notification messages, the receiver should be suspended with the reason "unsupportable-volume".

For configured subscriptions, operations are against the set of receivers using the subscription "id" as a handle for that set. But for streaming updates, subscription state change notifications are local to a receiver. In this specification it is the case that receivers get no information from the publisher about the existence of other receivers. But if a network operator wants to let the receivers correlate results, it is useful to use the subscription "id" across the receivers to allow that correlation. Note that due to the possibility of different access control permissions per receiver, each receiver may actually get a different set of event records.

For configured replay subscriptions, the receiver is protected from duplicated events being pushed after a publisher is rebooted. However it is possible that a receiver might want to acquire event records which failed to be delivered just prior to the reboot. Delivering these event records be accomplished by leveraging the "eventTime" from the last event record received prior to the receipt of a "subscription-started" subscription state change notification. With this "eventTime" and the "replay-start-time" from the "subscription-started" notification, an independent dynamic

subscription can be established which retrieves any event records which may have been generated but not sent to the receiver.

5.3. Transport Requirements

This section provides requirements for any subscribed notification transport supporting the solution presented in this document.

The transport selected by the subscriber to reach the publisher MUST be able to support multiple "establish-subscription" requests made within the same transport session.

For both configured and dynamic subscriptions the publisher MUST authenticate a receiver via some transport level mechanism before sending any event records for which they are authorized to see. In addition, the receiver MUST authenticate the publisher at the transport level. The result is mutual authentication between the two.

A secure transport is highly recommended. Beyond this, the publisher MUST ensure that the receiver has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

A specific transport specification built upon this document may or may not choose to require the use of the same logical channel for the RPCs and the event records. However the event records and the subscription state change notifications MUST be sent on the same transport session to ensure the properly ordered delivery.

A specific transport specification MUST identity any encoding supported. Where a configured subscription's transport allows different encodings, the specification MUST identify the default encoding.

Additional transport requirements will be dictated by the choice of transport used with a subscription. For an example of such requirements with NETCONF transport, see [I-D.draft-ietf-netconf-netconf-event-notifications].

5.4. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management transports such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest

RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF operations and content.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. To counter this, notification messages SHOULD NOT be sent to any receiver which does not support this specification. Receivers that do not want notification messages need only terminate or refuse any transport sessions from the publisher.

When a receiver of a configured subscription gets a new "subscription-started" message for a known subscription where it is already consuming events, it may indicate that an attacker has done something that has momentarily disrupted receiver connectivity. To acquire events lost during this interval, the receiver SHOULD retrieve any event records generated since the last event record was received. This can be accomplished by establishing a separate dynamic replay subscription with the same filtering criteria with the publisher, assuming the publisher supports the "replay" feature.

For dynamic subscriptions, implementations need to protect against malicious or buggy subscribers which may send a large number "establish-subscription" requests, thereby using up system resources. To cover this possibility operators SHOULD monitor for such cases and, if discovered, take remedial action to limit the resources used, such as suspending or terminating a subset of the subscriptions or, if the underlying transport is session based, terminate the underlying transport session.

The replay mechanisms described in Sections Section 2.4.2.1 and Section 2.5.6 provides access to historical event records. By design, the access control model that protects these records could enable subscribers to view data to which they were not authorized at the time of collection.

Using DNS names for configured subscription receiver "name" lookup can cause situations where the name resolves unexpectedly differently on the publisher, so the recipient would be different than expected.

An attacker that can cause the publisher to use an incorrect time can induce message replay by setting the time in the past, and introduce a risk of message loss by setting the time in the future.

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

Container: "/filters"

- "stream-subtree-filter": updating a filter could increase the computational complexity of all referencing subscriptions.

- "stream-xpath-filter": updating a filter could increase the computational complexity of all referencing subscriptions.

Container: "/subscriptions"

The following considerations are only relevant for configuration operations made upon configured subscriptions:

- "configured-replay": can be used to send a large number of event records to a receiver.

- "dependency": can be used to force important traffic to be queued behind less important updates.

- "dscp": if unvalidated, can result in the sending of traffic with a higher priority marking than warranted.

- "id": can overwrite an existing subscription, perhaps one configured by another entity.

- "name": adding a new key entry can be used to attempt to send traffic to an unwilling receiver.

- "replay-start-time": can be used to push very large logs, wasting resources.

- "source-address": the configured address might not be able to reach a desired receiver.

- "source-interface": the configured interface might not be able to reach a desired receiver.

- "source-vrf": can place a subscription into a virtual network where receivers are not entitled to view the subscribed content.

- "stop-time": could be used to terminate content at an inopportune time.

- "stream": could set a subscription to an event stream containing no content permitted for the targeted receivers.

- "stream-filter-name": could be set to a filter which is irrelevant to the event stream.

- "stream-subtree-filter": a complex filter can increase the computational resources for this subscription.

- "stream-xpath-filter": a complex filter can increase the computational resources for this subscription.

- "weighting": placing a large weight can overwhelm the dequeuing of other subscriptions.

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

Container: "/streams"

- "name": if access control is not properly configured, can expose system internals to those who should have no access to this information.

- "replay-support": if access control is not properly configured, can expose logs to those who should have no access.

Container: "/subscriptions"

- "excluded-event-records": leaf can provide information about filtered event records. A network operator should have permissions to know about such filtering. Improper configuration could provide a receiver with information leakage consisting of the dropping of event records.

- "subscription": different operational teams might have a desire to set varying subsets of subscriptions. Access control should be designed to permit read access to just the allowed set.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus

important to control access to these operations. These are the operations and their sensitivity/vulnerability:

RPC: all

- If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an excessive use of system resources on the publisher just to determine that these subscriptions should be declined. In such a situation, subscription interactions MAY be terminated by terminating the transport session.

RPC: "delete-subscription"

- No special considerations.

RPC: "establish-subscription"

- Subscriptions could overload a publisher's resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.

RPC: "kill-subscription"

- The "kill-subscription" RPC MUST be secured so that only connections with administrative rights are able to invoke this RPC.

RPC: "modify-subscription"

- Subscriptions could overload a publisher's resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.

6. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Martin Bjorklund, Kent Watsen, Balazs Lengyel, Robert Wilton, Sharon Chisholm, Hector Trevino, Susan Hares, Michael Scharf, and Guangying Zheng.

7. References

7.1. Normative References

[I-D.draft-ietf-rtgwg-ni-model]
Berger, L., Hopps, C., and A. Lindem, "YANG Network Instances", draft-ietf-rtgwg-ni-model-12 (work in progress), March 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>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

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

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

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[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>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[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., and A. Tripathy, "NETCONF support for event notifications", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>.

[I-D.draft-ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[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>.

[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>.

[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>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Example Configured Transport Augmentation

This appendix provides a non-normative example of how the YANG model defined in Section 4 may be enhanced to incorporate the configuration parameters needed to support the transport connectivity process. This example is not intended to be a complete transport model. In

this example, connectivity via an imaginary transport type of "foo" is explored. For more on the overall need, see Section 2.5.7.

The YANG model defined in this section contains two main elements. First is a transport identity "foo". This transport identity allows a configuration agent to define "foo" as the selected type of transport for a subscription. Second is a YANG case augmentation "foo" which is made to the "/subscriptions/subscription/receivers/receiver" node of Section 4. Within this augmentation are the transport configuration parameters "address" and "port" which are necessary to make the connect to the receiver.

```
module example-foo-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:example:foo-subscribed-notifications";

  prefix fsn;

  import ietf-subscribed-notifications {
    prefix sn;
  }
  import ietf-inet-types {
    prefix inet;
  }

  description
    "Defines 'foo' as a supported type of configured transport for
    subscribed event notifications.";

  identity foo {
    base sn:transport;
    description
      "Transport type 'foo' is available for use as a configured
       subscription transport protocol for subscribed notifications.";
  }

  augment
    "/sn:subscriptions/sn:subscription/sn:receivers/sn:receiver" {
    when 'derived-from(../../../transport, "fsn:foo")';
    description
      "This augmentation makes 'foo' specific transport parameters
      available  for a receiver.";
    leaf address {
      type inet:host;
      mandatory true;
      description
        "Specifies the address to use for messages destined to a
```

```
          receiver.";
      }
      leaf port {
        type inet:port-number;
        mandatory true;
        description
          "Specifies the port number to use for messages destined to a
          receiver.";
      }
    }
  }
```

Figure 21: Example Transport Augmentation for the fictitious protocol foo

This example YANG model for transport "foo" will not be seen in a real world deployment. For a real world deployment supporting an actual transport technology, a similar YANG model must be defined.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v24 - v25

- Replay security consideration added based on Roman Danyliw's discuss
- Spelling fixes, acronyms expanded
- Tweaks and updates based Benjamin Kaduk's comments. This includes the adding of clarifying security considerations, a couple of claifications in the YANG definitions, and ensuring a fuller set of transport specification requirements are defined in 5.3.

v23 - v24

- Per Benjamin Kaduk's discuss, adjusted IPR to pre5378Trust200902
- Tweaks from Chris Lonvick's IESG review. This includes moving a paragraph from Security Considerations into a sentence within Implementation Considerations.
- Tweaks from Wesley Eddy DSCP description

v22 - v23

- During the YANG Doctor review, feature dscp support was refined to avoid the out-of-order delivery of packets in a single TCP session.

v21 - v22

- YANG Dr definition clarifications. This includes refined text on: (a) stop-time can be used without replay, (b) a separate dynamic subscription for replay, (c) subscription state change notifications can't be dropped, more details on "enum concluded" and (d) more text on configurable-encoding leaf (which adds two informative references). There also was one minor tweak in the YANG model. The stream description leaf had "mandatory true" removed.

v20 - v21

- Editorial change in Section 1.3 requested by Qin's Shepherd review of NETCONF-Notif and RESTCONF-Notif. Basically extra text was added further describing that dynamic subscriptions can have state change notifications.

v18 - v20

- XPath-stream-filter YANG object definition updated based on NETMOD discussions.

v17 - v18

- Transport optional in YANG model.

- Modify subscription must come from the originator of the subscription. (Text got dropped somewhere previously.)

- Title change.

v16 - v17

- YANG renaming: Subscription identifier renamed to id. Counters renamed. Filters id made into name.

- Text tweaks.

v15 - v16

- Mandatory empty case "transport" removed.

- Appendix case turned from "netconf" to "foo".

v14 - v15

- Text tweaks.
- Mandatory empty case "transport" added for transport parameters. This includes a new section and an appendix explaining it.

v13 - v14

- Removed the 'address' leaf.
- Replay is now of type 'empty' for configured.

v12 - v13

- Tweaks from Kent's comments
- Referenced in YANG model updated per Tom Petch's comments
- Added leaf replay-previous-event-time
- Renamed the event counters, downshifted the subscription states

v11 - v12

- Tweaks from Kent's, Tim's, and Martin's comments
- Clarified dscp text, and made its own feature
- YANG model tweaks alphabetizing, features.

v10 - v11

- access control filtering of events in streams included to match RFC5277 behavior
- security considerations updated based on YANG template.
- dependency QoS made non-normative on HTTP2 QoS
- tree diagrams referenced for each figure using them
- reference numbers placed into state machine figures
- broke configured replay into its own section
- many tweaks updates based on LC and YANG doctor reviews

- trees and YANG model reconciled were deltas existed
- new feature for interface originated.
- dscp removed from the qos feature
- YANG model updated in a way which collapses groups only used once so that they are part of the 'subscriptions' container.
- alternative encodings only allowed for transports which support them.

v09 - v10

- Typos and tweaks

v08 - v09

- NMDA model supported. Non NMDA version at https://github.com/netconf-wg/rfc5277bis/
- Error mechanism revamped to match to embedded implementations.
- Explicitly identified error codes relevant to each RPC/ Notification

v07 - v08

- Split YANG trees to separate document subsections.
- Clarified configured state machine based on Balazs comments, and moved it into the configured subscription subsections.
- Normative reference to Network Instance model for VRF
- One transport for all receivers of configured subscriptions.
- QoS section moved in from yang-push

v06 - v07

- Clarification on state machine for configured subscriptions.

v05 - v06

- 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 event stream filters.
- Encoding refined in establish-subscription so it takes the RPC's encoding as the default.
- Namespaces in examples fixed.

v04 - v05

- Returned to the explicit filter subtyping of v00
- stream object changed to 'name' from 'stream'
- Cleaned up examples
- Clarified that JSON support needs notification-messages draft.

v03 - v04

- Moved back to the use of RFC5277 one-way notifications and encodings.

v03 - v04

- Replay updated

v02 - v03

- RPCs and Notification support is identified by the Notification 2.0 capability.
- Updates to filtering identities and text
- New error type for unsupportable volume of updates
- Text tweaks.

v01 - v02

- Subscription status moved under receiver.

v00 - v01

- Security considerations updated
- Intro rewrite, as well as scattered text changes
- Added Appendix A, to help match this to related drafts in progress
- Updated filtering definitions, and filter types in yang file, and moved to identities for filter types
- Added Syslog as an event stream
- HTTP2 moved in from YANG-Push as a transport option
- Replay made an optional feature for events. Won't apply to datastores
- Enabled notification timestamp to have different formats.
- Two error codes added.

v01 5277bis - v00 subscribed notifications

- Kill subscription RPC added.
- Renamed from 5277bis to Subscribed Notifications.
- Changed the notification capabilities version from 1.1 to 2.0.
- Extracted create-subscription and other elements of RFC5277.
- Error conditions added, and made specific in return codes.
- Simplified yang model structure for removal of 'basic' grouping.
- Added a grouping for items which cannot be statically configured.
- Operational counters per receiver.
- Subscription-id and filter-id renamed to identifier
- Section for replay added. Replay now cannot be configured.
- Control plane notification renamed to subscription state change 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
Microsoft

Email: alberto.gonzalez@microsoft.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com