NETCONF
Internet-Draft
Intended status: Standards Track
Expires: July 28, 2018

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
January 24, 2018

Custom Subscription to Event Streams
draft-ietf-netconf-subscribed-notifications-09

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on July 28, 2018.

Table of Contents

1. Introduction

This document defines capabilities and operations for the customized establishment of subscriptions upon system generated event streams. Effectively this enables a "subscribe then publish" capability where the customized information needs of each target receiver are understood by the publisher before subscribed event records are

marshaled and pushed. The receiver then gets a continuous, custom feed of publisher generated information.

While the functionality defined in this document is transport-agnostic, protocols like NETCONF [RFC6241] or RESTCONF [RFC8040] can be used to configure or dynamically signal subscriptions, and there are bindings defined for subscribed event record delivery for NETCONF within [I-D.draft-ietf-netconf-netconf-event-notifications], and for HTTP2 or HTTP1.1 within [I-D.draft-ietf-netconf-restconf-notif].

1.1. Motivation

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

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

1.2. Terminology

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

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

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

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

Event record: A set of information detailing an event.

NACM: NETCONF Access Control Model.

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

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

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

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

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

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

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

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

1.3. Solution Overview

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

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. If the publisher wants to serve this request, it accepts it, and then starts pushing notification messages. If the publisher does not

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

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

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bounded by the transport session used to establish it. For connection-oriented stateful transport like NETCONF, the loss of the transport session will result in the immediate termination of any associated dynamic subscriptions. For connectionless or stateless transports like HTTP, a lack of receipt acknowledgment of a sequential set of notification messages and/or keep-alives can be used to trigger a termination of a dynamic subscription. Contrast this to the lifetime of a configured subscription. This lifetime is driven by relevant configuration being present within the publisher's running configuration. Being tied to configuration operations implies configured subscriptions can be configured to persist across reboots, and implies a configured subscription can persist even when its publisher is fully disconnected from any network.

- Configured subscriptions can be modified by any configuration client with write permission on the configuration of the subscription. Dynamic subscriptions can only be modified via an RPC request made by the original subscriber.

Note that there is no mixing-and-matching of dynamic and configured operations on a single subscription. Specifically, a configured subscription cannot be modified or deleted using RPCs defined in this document. Similarly, a subscription established via RPC cannot be modified through configuration operations. Also note that transport specific transport drafts based on this specification MUST detail the life cycles of both dynamic and configured subscriptions.

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

1.4. Relationship to RFC-5277

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

- the data model in this document replaces the data model in [RFC5277].
- the RPC operations in this draft replaces the symmetrical operations of [RFC5277], section 4.
- the one way operation of [RFC5277] is still used. However this operation will no longer be required with the availability of [I.D.draft-ietf-netconf-notification-messages].
- the definition and contents of the NETCONF stream are identical between this document and [RFC5277].
- a publisher MAY implement both the data model and RPCs defined in [RFC5277] and this new document concurrently, in order to support old clients. However the use of both alternatives on a single transport session is prohibited.

2. Solution

2.1. Event Streams

An event stream is a named entity on a publisher which exposes a continuously updating set of event records. Each event stream is available for subscription. It is out of the scope of this document to identify a) how streams are defined, b) how event records are defined/generated, and c) how event records are assigned to streams.

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

As event records are created by a system, they may be assigned to one or more streams. The event record is distributed to subscription's

receiver(s) where: (1) a subscription includes the identified stream, and (2) subscription filtering does not exclude the event record from that receiver.

If access control permissions are in use to secure publisher content, then for event records to be sent to a receiver, that receiver MUST be allowed access to all the event records on the stream. If subscriber permissions change during the lifecycle of a subscription, then the subscription MUST be continued or terminated accordingly.

2.2. Event Stream Filters

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

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

2.3. QoS

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

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

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

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

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

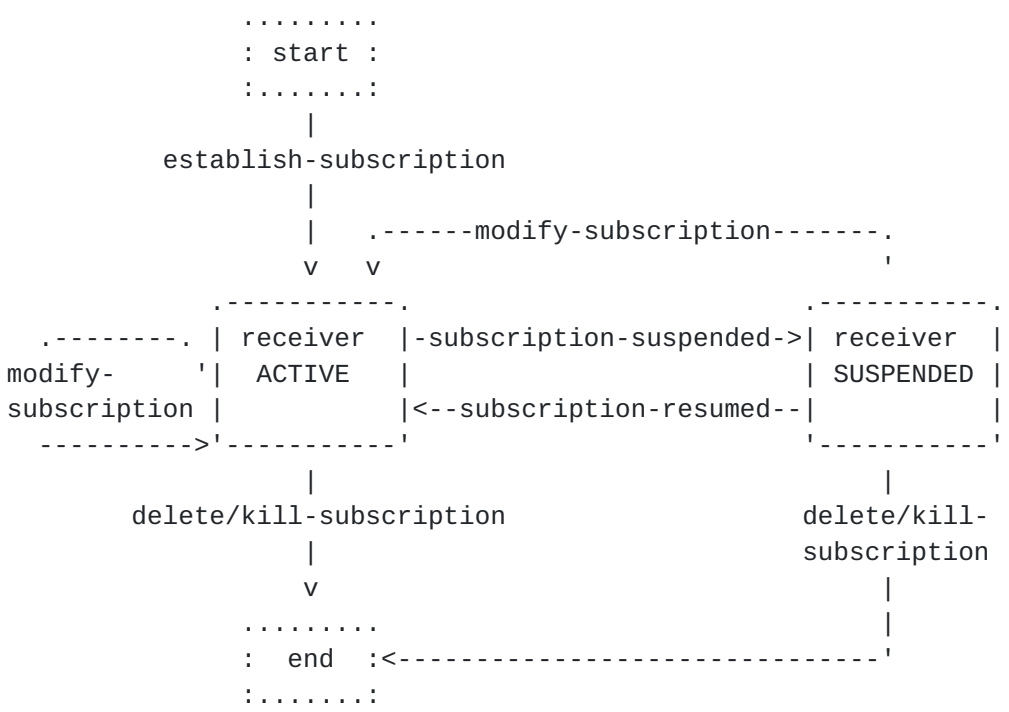
A subscription's dependency MUST work identically to stream dependency as described within [RFC7540], sections 5.3.1, 5.3.3, and 5.3.4. If a dependency is attempted via an RPC, but the referenced subscription does not exist, the dependency will be silently removed.

2.4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC, and are made against targets located within the publisher. These RPCs have been designed extensibly so that they may be augmented for subscription targets beyond event streams.

2.4.1. Dynamic Subscription State Model

Below is the publisher's state machine for a dynamic subscription. It is important to note that such a subscription doesn't exist at the publisher until an "establish-subscription" RPC is accepted. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible. States that are reflected in the YANG model appear in upper-case letters; in addition, start and end states are depicted to reflect subscription creation and deletion events.



Receiver state for a dynamic subscription

Of interest in this state machine are the following:

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

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

- A delete or kill RPC will end the subscription.

- Suspend and resume state changes are driven by internal process and prioritization. There are no direct controls over suspend and resume other than modifying a subscription

2.4.2. Establishing a Subscription

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

The input parameters of the operation are:

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

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

- An optional encoding for the event records pushed. Note: If no encoding is included, the encoding of the RPC MUST be used.

- An optional stop time for the subscription. If no stop-time is present, notification messages will continue to be sent until the subscription is terminated.

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

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

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

Figure 1: establish-subscription RPC

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

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

Figure 2: establish-subscription RPC yang-data

2.4.2.1. Replay Subscription

Replay provides the ability to establish a subscription which is also capable of passing recently generated event records. In other words, as the subscription initializes itself, it sends any previously generated content from within target event stream which meets the filter and timeframe criteria. These historical event records would

then be followed by event records generated after the subscription has been established. All event records will be delivered in the order generated.

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

The inclusion of a replay-start-time within an "establish-subscription" RPC indicates a replay request. If the "replay-start-time" contains a value that is earlier than content stored within the publisher's replay buffer, then the subscription MUST be rejected, and the leaf "replay-start-time-hint" MUST be set in the reply.

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

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

If a stream supports replay, the "replay-support" leaf is present in the "/streams/stream" list entry for the stream. An event stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given replay request. To assess the availability of replay, subscribers can perform a get on "replay-log-creation-time" and "replay-log-aged-time". See Figure 10 for the tree describing these elements. The actual size of the replay log at any given time is a publisher specific matter. Control parameters for the replay log are outside the scope of this document.

2.4.3. Modifying a Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription previously established on that transport session via "establish-subscription". Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the requested modifications, it acknowledges success to the subscriber, then immediately starts sending event records based on the new terms.

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

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

Figure 3: modify-subscription RPC

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

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

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

Figure 4: modify-subscription RPC yang-data

2.4.4. Deleting a Subscription

The "delete-subscription" operation permits canceling an existing subscription previously established on that transport session. If the publisher accepts the request, and the publisher has indicated success, the publisher MUST NOT send any more notification messages

for this subscription. If the publisher rejects the request, the request has no impact whatsoever on any subscription.

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

Figure 5: delete-subscription RPC

Dynamic subscriptions can only be deleted via this RPC using the same transport session previously used for subscription establishment. Configured subscriptions cannot be deleted using RPCs.

2.4.5. Killing a Subscription

The "kill-subscription" operation permits an operator to end a dynamic subscription which is not associated with the transport session used for the RPC. This operation MUST be secured so that only connections with sufficiently privileged access rights are able to invoke this RPC. A publisher MUST terminate any dynamic subscription identified by RPC request. An operator may find subscription identifiers which may be used with "kill-subscription" by searching for the IP address of a receiver within "subscriptions\subscription\receivers\receiver\address".

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

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

2.4.6. RPC Failures

Whenever an RPC is unsuccessful, the publisher returns relevant error codes as part of the RPC error response. RPC error codes returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241] Appendix A, as well as subscription specific errors such as those defined within this document's YANG model. As a result of this mixture, how subscription errors are encoded within an RPC error response is transport dependent.

There are elements of the RPC error mechanism which are transport independent. Specifically, references to specific identities within the YANG model MUST be returned as part of the error responses

resulting from failed attempts at event stream subscription. Following are valid errors per RPC:

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

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

There is one final set of transport independent RPC error elements included in the YANG model. These are the following three yang-data structures for failed event stream subscriptions:

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

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

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

2.5. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface. Configured subscriptions may be modified by any configuration client with the proper permissions. Subscriptions can be modified or terminated via the configuration interface at any point of their lifetime.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions:

- persistence across reboots,

- persistence even when transport is unavailable, and

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

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

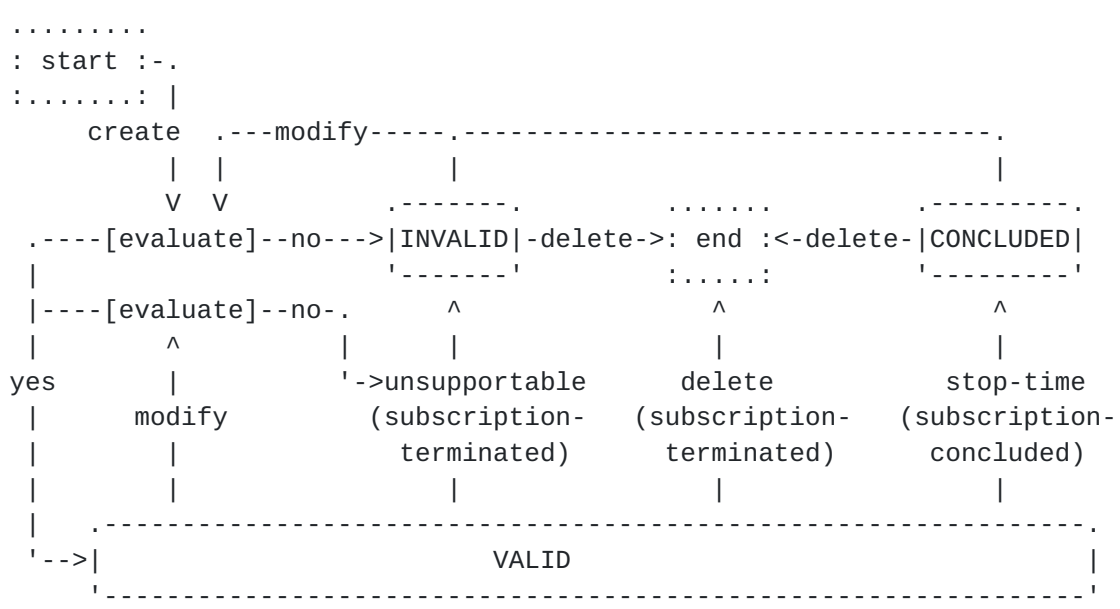
In addition to subscription parameters available to dynamic subscriptions as described in Section 2.4.2, the following additional parameters are also available to configured subscriptions:

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

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

2.5.1. Configured Subscription State Model

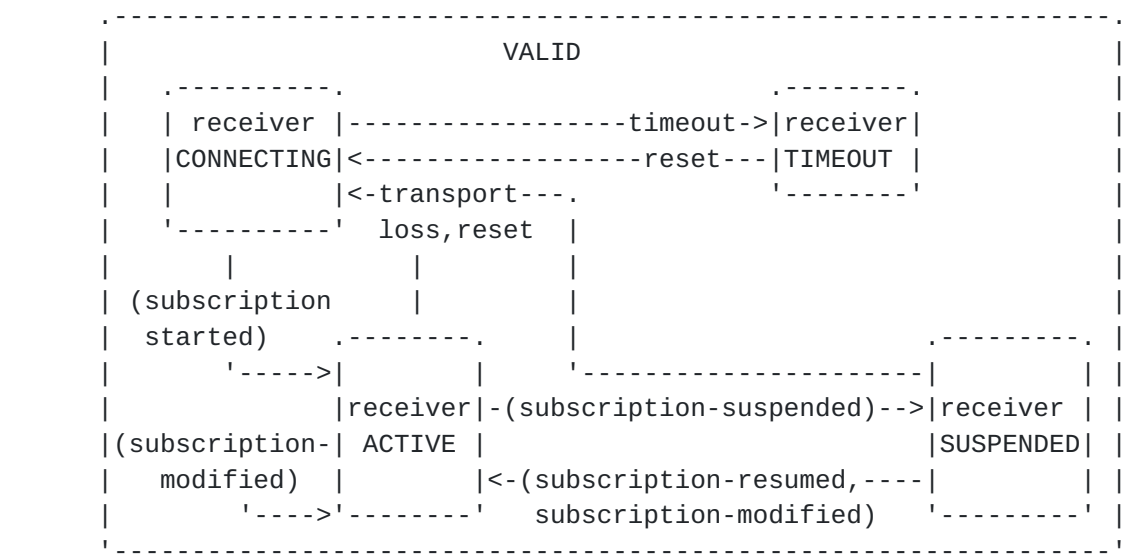
Below is the state machine for a configured subscription. States that are reflected in the YANG model appear in upper-case letters; in addition, start and end states are depicted to reflect configured subscription creation and deletion events. The creation or modification of a configured subscription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



State model for a configured subscription.

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

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



Receiver state for a configured subscription

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

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

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

The mechanisms described above is mirrored in the RPCs and YANG notifications within the document. It should be noted that these RPCs and YANG notifications have been designed to be extensible and

allow subscriptions into targets other than event streams. [I-D.ietf-netconf-yang-push] provides an example of such an extension.

2.5.2. Creating a Configured Subscription

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

After a subscription is successfully created, the publisher immediately sends a "subscription-started" state change notification to each receiver. It is quite possible that upon configuration, reboot, or even steady-state operations, a transport session may not be currently available to the receiver. In this case, when there is something to transport for an active subscription, transport specific call-home operations will be used to establish the connection. When transport connectivity is available, notification messages may then be pushed.

With active configured subscriptions, it is allowable to buffer event records even after a "subscription-started" has been sent. However if events are lost (rather than just delayed) due to replay buffer overflow, a new "subscription-started" must be sent. This new "subscription-started" indicates an event record discontinuity.

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

Note that is possible to configure replay on a configured subscription. This capability is to allow a configured subscription to exist on a system so that event records generated during boot can be buffered and pushed as soon as the transport session is established.

2.5.3. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree "subscription-config".

If the modification involves adding receivers, added receivers are placed in the "connecting" state. If a receiver is removed, the state change notification "subscription-terminated" is sent to that receiver if that receiver is "active" or "suspended" .

If the modification involved changing the policies for the subscription, the publisher sends to currently active receivers a "subscription-modified" notification. For any suspended receivers, a "subscription-modified" notification will be delayed until the receiver is resumed. (Note: in this case, the "subscription-modified" notification informs the receiver that the subscription has been resumed, so no additional "subscription-resumed" need be sent.)

2.5.4. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree "subscription-config".

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

2.5.5. Resetting a Configured Receiver

It is possible that a configured subscription to a receiver needs to be reset. This re-initialization may be useful in cases where a publisher has timed out trying to reach a receiver. When such a reset occurs, a transport session will be initiated if necessary, and a new "subscription-started" notification will be sent.

2.6. Event Record Delivery

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

A notification message is sent to a receiver when an event record is able to traverse the specified filter criteria. This notification

message MUST be encoded as one-way notification element of [RFC5277], Section 4. The following example within [RFC7950] section 7.16.3 is an example of a compliant message:

```
<notification
       xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <link-failure xmlns="http://acme.example.com/system">
        <if-name>so-1/2/3.0</if-name>
        <if-admin-status>up</if-admin-status>
        <if-oper-status>down</if-oper-status>
    </link-failure>
</notification>
```

Figure 6: subscribed notification message

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

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

2.7. Subscription State Notifications

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

Subscription state notifications are unlike other notifications which might be found in the event stream. They cannot be filtered out, and they are delivered only to directly impacted receiver(s) of a subscription. The identification of subscription state notifications is easy to separate from other notification messages through the use of the YANG extension "subscription-state-notif". This extension tags a notification as subscription state notification.

The complete set of subscription state notifications is described in the following subsections.

2.7.1. subscription-started

This notification indicates that a configured subscription has started, and event records may be sent. Included in this state change notification are all the parameters of the subscription, except for the receiver(s) addressing information and origin information indicating where notification messages will egress the publisher. Note that if a referenced filter from the "filters" container has been used within the subscription, the notification will include the contents of that referenced under the "within-subscription" subtree.

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

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

Figure 7: subscription-started notification

2.7.2. subscription-modified

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

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

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

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

2.7.3. subscription-terminated

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

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

Figure 8: subscription-terminated notification

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

2.7.4. subscription-suspended

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

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

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

2.7.5. subscription-resumed

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

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

Figure 9: subscription-resumed notification

2.7.6. subscription-completed

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

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

2.7.7. replay-completed

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

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

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

2.8. Subscription Monitoring

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

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

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

2.9. Advertisement

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

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

3. YANG Data Model Trees

This section contains tree diagrams for top level YANG Data Node containers defined in Section 4. If you would rather see tree diagrams for Notifications, see Section 2.7. Or for the tree diagrams for the RPCs, see Section 2.4.

3.1. Event Streams Container

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. The list of event streams that are supported by the publisher and against which subscription is allowed may be acquired from the "streams" container within the YANG module.

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

Figure 10: Stream Container

3.2. Event Stream Filters Container

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

```
    +--rw filters
       +--rw stream-filter* [identifier]
          +--rw identifier               filter-id
          +--rw (filter-spec)?
             +--:(stream-subtree-filter)
             |  +--rw stream-subtree-filter?    {subtree}?
             +--:(stream-xpath-filter)
                +--rw stream-xpath-filter?     yang:xpath1.0 {xpath}?
```

Figure 11: Filter Container

3.3. Subscriptions Container

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

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

4. Data Model

```
<CODE BEGINS> file "ietf-subscribed-notifications@2018-01-25.yang"
module ietf-subscribed-notifications {
  yang-version 1.1;
```

```
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

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

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

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

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

     Editor:   Alberto Gonzalez Prieto
               <mailto:agonzalezpri@vmware.com>

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

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

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

  revision 2018-01-25 {
    description
      "Initial version";
```

```
    reference
      "draft-ietf-netconf-subscribed-notifications-09";
  }

  /*
   * FEATURES
   */

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

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

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

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

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

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

  feature supports-vrf {
    description
      "This feature indicates a publisher supports VRF configuration
```

```
      for configured subscriptions.  VRF support for dynamic
      subscriptions does not require this feature.";
    reference "draft-ietf-rtgwg-ni-model";
  }

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

  /*
   * EXTENSIONS
   */

  extension subscription-state-notification {
    description
      "This statement applies only to notifications. It indicates that
       the notification is a subscription state notification. Therefore
       it does not participate in a regular event stream and does not
       need to be specifically subscribed to in order to be received.
       This statement can only occur as a substatement to the YANG
       'notification' statement.";
  }

  /*
   * IDENTITIES
   */

  /* Identities for RPC and Notification errors */

  identity establish-subscription-error {
     description
      "Problem found while attempting to fulfill an
      'establish-subscription' rpc request. ";
  }

  identity modify-subscription-error {
     description
      "Problem found while attempting to fulfill a
      'modify-subscription' rpc request. ";
  }

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

```
      rpc request. ";
  }

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

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

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

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

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

  identity history-unavailable {
    base establish-subscription-error;
    description
     "Replay request too far into the past. This means the publisher
      does store historic information for the requested stream, but
      not back to the requested timestamp.";
  }

  identity insufficient-resources {
    base establish-subscription-error;
```

```
    base modify-subscription-error;
    base subscription-suspended-reason;
    description
      "The publisher has insufficient resources to support the
       requested subscription.";
  }

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

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

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

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

  identity unsupportable-volume {
    base subscription-suspended-reason;
    description
      "The publisher cannot support the volume of information intended
      to be sent for an existing subscription.";
  }

  /* Identities for encodings */
```

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

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

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

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

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

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

  identity http1.1 {
    base transport;
    description
      "HTTP1.1 is used a transport for notification messages and state
       change notifications.";
```

```
    reference "draft-ietf-netconf-restconf-notif-03, Section 3.1.2";
  }

  /*
   * TYPEDEFs
   */

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

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

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

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

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

  typedef stream-filter-ref {
    type leafref {
      path "/sn:filters/sn:stream-filter/sn:identifier";
```

```
    }
    description
      "This type is used to reference a configured stream filter.";
  }

  /*
   * GROUPINGS
   */

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

          The subtree filter is applied to the representation of
          individual, delineated event records as contained within the
          event stream.  For example, if the notification message
          contains an instance of a notification defined in YANG, then
          the top-level element is the name of the YANG notification.

          If the subtree filter returns a non-empty node set, the filter
          matches the event record, and the it is included in the
          notification message sent to the receivers.";
        reference "RFC 6241, Section 6.";
      }
      leaf stream-xpath-filter {
        if-feature "xpath";
        type yang:xpath1.0;
        description
          "Event stream evaluation criteria encoded in the syntax of
           an XPath 1.0 expression.

           The XPath expression is evaluated on the representation of
           individual, delineated event records as contained within
           the event stream.  For example, if the notification message
           contains an instance of a notification defined in YANG,
           then the top-level element is the name of the YANG
           notification, and the root node has this top-level element
           as the only child.
```

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

          The expression is evaluated in the following XPath context:

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

            o  The set of variable bindings is empty.

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

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

      }
    }
  }

  grouping update-qos {
    description
      "This grouping describes Quality of Service information
       concerning a subscription.  This information is passed to lower
       layers for transport prioritization and treatment";
    leaf dscp {
      if-feature "qos";
      type inet:dscp;
      default "0";
      description
        "The push update's IP packet transport priority. This is made
         visible across network hops to receiver. The transport
         priority is shared for all receivers of a given subscription.";
    }
    leaf weighting {
      if-feature "qos";
      type uint8 {
         range "0 .. 255";
      }
      description
        "Relative weighting for a subscription. Allows an underlying
         transport layer perform informed load balance allocations
         between various subscriptions";
```

```
      reference
        "RFC-7540, section 5.3.2";
    }
    leaf dependency {
      if-feature "qos";
      type subscription-id;
      description
        "Provides the Subscription ID of a parent subscription which
         has absolute priority should that parent have push updates
         ready to egress the publisher. In other words, there should be
         no streaming of objects from the current subscription if
         the parent has something ready to push.";
      reference
        "RFC-7540, section 5.3.1";
    }
  }

  grouping subscription-policy-modifiable {
    description
      "This grouping describes all objects which may be changed
      in a subscription via an RPC.";
    choice target {
      mandatory true;
      description
        "Identifies the source of information against which a
        subscription is being applied, as well as specifics on the
        subset of information desired from that source.  This choice
        exists so that additional filter types can be added via
        augmentation.";
      case stream {
        choice stream-filter {
          description
            "An event stream filter can be applied to a subscription.
            That filter will come either referenced from a global list,
            or be provided within the subscription itself.";
          case by-reference {
            description
              "Apply a filter that has been configured separately.";
            leaf stream-filter-ref {
              type stream-filter-ref;
              mandatory true;
              description
                "References an existing stream-filter which is to
                be applied to stream for the subscription.";
            }
          }
          case within-subscription {
            description
```

```
              "Local definition allows a filter to have the same
              lifecycle as the subscription.";
            uses stream-filter-elements;
          }
        }
      }
    }
    leaf stop-time {
      type yang:date-and-time;
      description
        "Identifies a time after which notification messages for a
        subscription should not be sent.  If stop-time is not present,
        the notification messages will continue until the subscription
        is terminated.  If replay-start-time exists, stop-time must be
        for a subsequent time. If replay-start-time doesn't exist,
        stop-time must be for a future time.";
    }
  }

  grouping subscription-policy-dynamic {
    description
      "This grouping describes information concerning a subscription
      which can be passed over the RPCs defined in this model.";
    leaf encoding {
      type encoding;
      mandatory true;
      description
        "The type of encoding for the subscribed data.";
    }
    uses subscription-policy-modifiable {
      augment target/stream {
        description
          "Adds additional objects which can be modified by RPC.";
        leaf stream {
          type stream-ref {
            require-instance false;
          }
          mandatory true;
          description
            "Indicates the stream of event records to be considered for
            this subscription.";
        }
        leaf replay-start-time {
          if-feature "replay";
          type yang:date-and-time;
          description
            "Used to trigger the replay feature and indicate that the
            replay should start at the time specified.  If
```

```
            replay-start-time is not present, this is not a replay
            subscription and event record push should start immediately.
            It is never valid to specify start times that are later than
            or equal to the current time.";
        }
      }
    }
    uses update-qos;
  }

  grouping subscription-policy {
    description
      "This grouping describes the full set of policy information
      concerning both dynamic and configured subscriptions, except for
      configured receivers.";
    leaf protocol {
      if-feature "configured";
      type transport;
      mandatory true;
      description
        "This leaf specifies the transport protocol used to deliver
        messages destined to all receivers of a subscription.";
    }
    uses subscription-policy-dynamic;
  }

  grouping notification-origin-info {
    description
      "Defines the sender source from which notification messages for a
       configured subscription are sent.";
    choice notification-message-origin {
      description
        "Identifies the egress interface on the Publisher from which
         notification messages are to be sent.";
      case interface-originated {
        description
          "When notification messages to egress a specific, designated
           interface on the Publisher.";
        leaf source-interface {
          type if:interface-ref;
          description
            "References the interface for notification messages.";
        }
      }
      case address-originated {
        description
          "When notification messages are to depart from a publisher
             using specfic originating address and/or routing context
```

```
           information.";
        leaf source-vrf {
          if-feature "supports-vrf";
          type leafref {
            path "/ni:network-instances/ni:network-instance/ni:name";
          }
          description
            "VRF from which notification messages should egress a
            publisher.";
        }
        leaf source-address {
          type inet:ip-address-no-zone;
          description
            "The source address for the notification messages.  If a
            source VRF exists, but this object doesn't, a publisher's
            default address for that VRF must be used.";
        }
      }
    }
  }

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

```
        }
      }
    }
  }


  /*
   * RPCs
   */

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


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

```
          base establish-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be created to a targeted stream.";
        }
      leaf filter-failure-hint {
        type string;
          description
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription.";
      }
      leaf replay-start-time-hint {
        type yang:date-and-time;
          description
            "If a replay has been requested, but the requested replay
            time cannot be honored, this may provide a hint at an
            alternate time which may be supportable.";
      }
    }
  }

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

  rc:yang-data modify-subscription-error-stream {
    container modify-subscription-error-stream {
```

```
      description
        "This yang-data MAY be provided as part of a subscription's RPC
        error response when there is a failure of a
        'modify-subscription' RPC which has been made against a
        stream.  This yang-data MUST be used if hints are to be
        provides back to the subscriber.";
      leaf reason {
        type identityref {
          base modify-subscription-error;
        }
        description
          "Information in a modify-subscription RPC error response which
          indicates the reason why the subscription to an event stream
          has failed to be modified.";
      }
      leaf filter-failure-hint {
        type string;
          description
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription.";
      }
    }
  }

  rpc delete-subscription {
    description
      "This RPC allows a subscriber to delete a subscription that
       was previously created from by that same subscriber using the
       establish-subscription RPC.";
    input {
      leaf identifier {
        type subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted.
           Only subscriptions that were created using
           establish-subscription can be deleted via this RPC.";
      }
    }
  }

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

```
        type subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted. Only
            subscriptions that were created using establish-subscription
           can be deleted via this RPC.";
      }
    }
  }

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

  /*
   * NOTIFICATIONS
   */

  notification replay-completed {
    sn:subscription-state-notification;
    if-feature "replay";
    description
      "This notification is sent to indicate that all of the replay
        notifications have been sent. It must not be sent for any other
       reason.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
```

```
  }

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

  notification subscription-started {
    sn:subscription-state-notification;
    if-feature "configured";
    description
      "This notification indicates that a subscription has started and
        notifications are beginning to be sent. This notification shall
       only be sent to receivers of a subscription; it does not
       constitute a general-purpose notification.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy {
      refine "target/stream/replay-start-time" {
         description
           "Indicates the time that a replay using for the streaming of
           buffered event records.  This will be populated with the most
           recent of the following: replay-log-creation-time,
           replay-log-aged-time, replay-start-time, or the most recent
           publisher boot time.";
      }
      refine "target/stream/stream-filter/within-subscription" {
        description
          "Filter applied to the subscription.  If the
          'stream-filter-ref' is populated, the filter within the
          subscription came from the 'filters' container.  Otherwise it
          is populated in-line as part of the subscription.";
      }
    }
  }
```

```
  notification subscription-resumed {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription that had
       previously been suspended has resumed. Notifications will once
       again be sent.  In addition, a subscription-resumed indicates
       that no modification of parameters has occurred since the last
       time event records have been sent.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification subscription-modified {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription has been
       modified.  Notification messages sent from this point on will
       conform to the modified terms of the subscription.  For
       completeness, this state change notification includes both
       modified and non-modified aspects of a subscription.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy {
      refine "target/stream/stream-filter/within-subscription" {
        description
          "Filter applied to the subscription.  If the
          'stream-filter-ref' is populated, the filter within the
          subscription came from the 'filters' container.  Otherwise it
          is populated in-line as part of the subscription.";
      }
    }
  }

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

```
      mandatory true;
      description
        "This references the affected subscription.";
    }
    leaf reason {
      type identityref {
        base subscription-terminated-reason;
      }
      mandatory true;
      description
        "Identifies the condition which resulted in the termination .";
    }
  }

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

  /*
   * DATA NODES
   */

  container streams {
    config false;
    description
      "This container contains information on the built-in streams
      provided by the publisher.";
```

```
    list stream {
      key "name";
      description
        "Identifies the built-in streams that are supported by the
         publisher.";
      leaf name {
        type string;
        description
          "A handle for a system-provided datastream made up of a
          sequential set of event records, each of which is
          characterized by its own domain and semantics.";
      }
      leaf description {
        type string;
        mandatory true;
        description
          "A description of the event stream, including such information
           as the type of event records that are available within this
           stream.";
      }
      leaf replay-support {
        if-feature "replay";
        type empty;
        description
          "Indicates that event record replay is available on this
          stream.";
      }
      leaf replay-log-creation-time {
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp of the creation of the log used to support the
          replay function on this stream. Note that this might be
          earlier then the earliest available information contained in
          the log. This object is updated if the log resets for some
          reason. This object MUST be present if replay is supported.";
      }
      leaf replay-log-aged-time {
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp of the last event record aged out of the log.
          This object MUST be present if replay is supported and any
          event record have been aged out of the log.";
      }
    }
  }
```

```
  container filters {
    description
      "This container contains a list of configurable filters
       that can be applied to subscriptions.  This facilitates
       the reuse of complex filters once defined.";
    list stream-filter {
      key "identifier";
      description
        "A list of pre-positioned filters that can be applied to
        subscriptions.";
      leaf identifier {
        type filter-id;
        description
          "An identifier to differentiate between filters.";
      }
      uses stream-filter-elements;
    }
  }

  container subscriptions {
    description
      "Contains the list of currently active subscriptions, i.e.
       subscriptions that are currently in effect, used for subscription
       management and monitoring purposes. This includes subscriptions
       that have been setup via RPC primitives as well as subscriptions
       that have been established via configuration.";
    list subscription {
      key "identifier";
      description
        "The identity and specific parameters of a subscription.
         Subscriptions within this list can be created using a control
         channel or RPC, or be established through configuration.";
      leaf identifier {
        type subscription-id;
        description
          "Identifier of a subscription; unique within a publisher";
      }
      leaf configured-subscription-state {
        if-feature "configured";
        type enumeration {
          enum valid {
            value 1;
            description
              "Connection is active and healthy.";
          }
          enum invalid {
            value 2;
            description
```

```
            "The subscription as a whole is unsupportable with its
            current parameters.";
          }
          enum concluded {
            value 3;
            description
              "A subscription is inactive as it has hit a stop time,
              but not yet been removed from configuration.";
          }
        }
        config false;
        description
          "The presence of this leaf indicates that the subscription
          originated from configuration, not through a control channel
          or RPC.  The value indicates the system established state
          of the subscription.";
      }
      leaf purpose {
        if-feature "configured";
        type string;
        description
          "Open text allowing a configuring entity to embed the
          originator or other specifics of this subscription.";
      }
      uses subscription-policy {
        refine "target/stream/stream" {
          description
            "Indicates the stream of event records to be considered for
            this subscription.   If a stream has been removed, and no
            longer can be referenced by an active subscription, send a
            'subscription-terminated' notification with
            'stream-unavailable' as the reason.  If a configured
            subscription refers to a non-existent stream, move that
            subscription to the 'invalid' state.";
        }
      }
      uses notification-origin-info {
        if-feature "configured";
      }
      uses receiver-info {
        augment receivers/receiver {
          description
            "include operational data for receivers.";
          leaf pushed-notifications {
            type yang:counter64;
            config false;
            description
              "Operational data which provides the number of update
```

```
            notification messages pushed to a receiver.";
        }
      leaf excluded-notifications {
        type yang:counter64;
        config false;
        description
          "Operational data which provides the number of event
           records from a stream explicitly removed via filtering so
           that they are not sent to a receiver.";
        }
      leaf state {
        type enumeration {
          enum active {
            value 1;
            description
              "Receiver is currently being sent any applicable
              notification messages for the subscription.";
          }
          enum suspended {
            value 2;
            description
              "Receiver state is suspended, so the publisher
              is currently unable to provide notification messages
              for the subscription.";
          }
          enum connecting {
            value 3;
            if-feature "configured";
            description
              "A subscription has been configured, but a
              subscription-started state change notification needs
              to be successfully received before notification
              messages are sent.";
          }
          enum timeout {
            value 4;
            if-feature "configured";
            description
              "A subscription has failed in sending a subscription
              started state change to the receiver.
              Additional attempts at connection attempts are not
              currently being made.";
          }
        }
        config false;
        mandatory true;
        description
          "Specifies the state of a subscription from the
```

```
              perspective of a particular receiver.  With this info it
              is possible to determine whether a subscriber is currently
              generating notification messages intended for that
              receiver.";
          }
          action reset {
            description
              "Allows the reset of this configured subscription receiver
              to the 'connecting' state. This enables the
              connection process to be reinitiated.";
            output {
              leaf time {
                type yang:date-and-time;
                mandatory true;
                description
                  "Time a publisher returned the receiver to a
                  connecting state.";
              }
            }
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

5. Considerations

5.1. Implementation Considerations

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

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

An implementation may choose to transition between active and suspended subscription states more frequently than required by this

specification. However if a subscription is unable to marshal all intended updates into a transmittable message in multiple successive intervals, the subscription SHOULD be suspended with the reason "unsupportable-volume".

For configured subscriptions, operations are against the set of receivers using the subscription identifier as a handle for that set. But for streaming up dates, state change notifications are local to a receiver. In this specification it is the case that receivers get no information from the publisher about the existence of other receivers. But if an operator wants to let the receivers correlate results, it is useful to use the subscription identifier handle across the receivers to allow that correlation.

5.2. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

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

5.3. Security Considerations

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

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

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

If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an excessive use of system resources. In such a situation, subscription interactions MAY be terminated by terminating the transport session.

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

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

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

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

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

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

One subscription id can be used for two or more receivers of the same configured subscription. But due to the possibility of different access control permissions per receiver, it SHOULD NOT be assumed that each receiver is getting identical updates.

6. Acknowledgments

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

7. References

7.1. Normative References

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

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

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

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

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

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

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

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

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

7.2. Informative References

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

[I-D.draft-ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", August 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", December 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

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

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

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

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

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v08 - v09

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

v07 - v08

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

v06 - v07

- Clarification on state machine for configured subscriptions.

v05 - v06

- Made changes proposed by Martin, Kent, and others on the list. Most significant of these are Stream returned to string (with the SYSLOG identity removed), intro section on 5277 relationship, an identity set moved to an enumeration, clean up of definitions/ terminology, state machine proposed for configured subscriptions with a clean-up of subscription state options.
- JSON and XML become features. Also Xpath and subtree filtering become features
- Terminology updates with event records, and refinement of filters to just stream filters.
- Encoding refined in establish-subscription so it takes the RPC's encoding as the default.
- Namespaces in examples fixed.

v04 - v05

- 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 a stream
- HTTP2 moved in from YANG-Push as a transport option

- Replay made an optional feature for events. Won't apply to datastores
- Enabled notification timestamp to have different formats.
- Two error codes added.

v01 5277bis - v00 subscribed notifications

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

v00 - v01 of 5277bis

- YANG Model changes. New groupings for subscription info to allow restriction of what is changeable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.
- Expanded/renamed definitions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.

- Removal of redundancy with other drafts
- Many other clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Alberto Gonzalez Prieto
VMWare

Email: agonzalezpri@vmware.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com