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

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
October 30, 2017

Custom Subscription to Event Streams
draft-ietf-netconf-subscribed-notifications-07

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 May 3, 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 marshalled 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 RPC subscription state signaling messages.

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 include event records.

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 a 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 associated dynamic subscriptions. For connectionless or stateless transports like HTTP, it is the lack of receipt acknowledgement of a sequential set of notification messages and/or keep-alives which will terminate dynamic subscriptions. The lifetime of a configured subscription is driven by relevant configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persist even when transport is unavailable.

- 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 upon the original subscribing transport session.

Note that there is no mixing-and-matching of dynamic and configured subscriptions. Specifically, a configured subscription cannot be modified or deleted using RPC. Similarly, a subscription established via RPC cannot be modified through configuration operations (if needed, an operator may use a kill RPC however).

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 either configured or dynamic subscriptions. Such termination or suspension MAY be driven by the publisher running out of resources to serve the subscription, or by internal errors on 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 realtionships 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 symetrical 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 events. 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 events are defined/ generated, and c) how events 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. Beyond NETCONF, implementations are free to add additional event streams.

As events are raised by a system, they may be assigned to one or more streams. The event record is distributed to receivers 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 notification messages 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]. The subsets of these filtering syntaxes supported are left to each implementation. A subset of information is never stripped from within the event record.

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

2.3. Subscription State Model at the Publisher

Below is the state machine of a subscription for the publisher for a dynamic subscription. It is important to note that such a subscription doesn't exist at the publisher until it is accepted and made active. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible via this state machine.

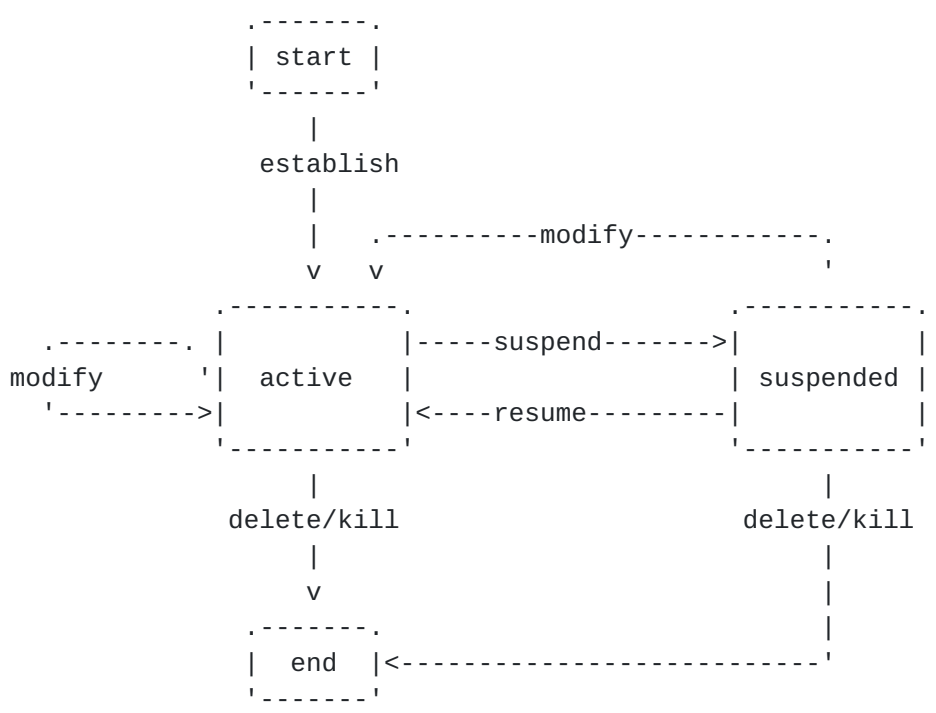

Figure 1: Dynamic subscription states

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 external controls over suspend and resume.

As shown below, a very similar state machine exists for configured subscriptions. Creation, modification, and deletion is via configuration operations rather than via RPC. When a subscription is first created, the operational status of each receiver is initially set to connecting. Individual are receivers are moved to an active status when a subscription-started state change notification is successfully delivered. Configured subscriptions remain active if transport connectivity is not lost and event records are not being dropped due to buffer overflow. A configured subscription MUST be moved connecting if transport connectivity is lost.

A configured subscription MUST be moved to a suspended status if notification messages are being dropped despite the presence of transport connectivity. A configured subscription MUST be returned to an active status from the suspended status as soon as transport connectivity is re-established, and a receiver acknowledges receipt of a "subscription-resumed". Otherwise, it MUST be moved to connecting.

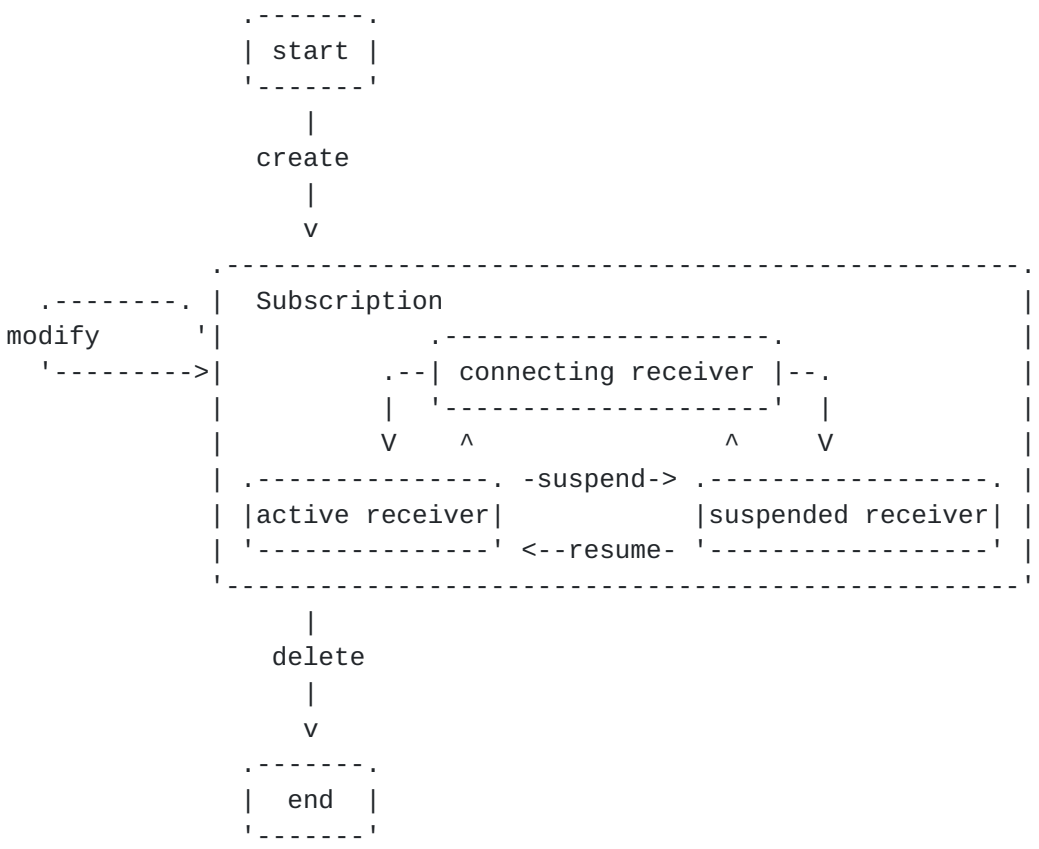

Figure 2: Configured subscription and receiver states

The interaction model described within this section is mirrored in the RPCs and Notifications later in the document. It should be noted that these RPCs and Notifications have been designed to be extensible and allow subscriptions into targets other than event streams. [I-D.ietf-netconf-yang-push] provides an example of such an extension.

3. Data Model Trees

```
module: ietf-subscribed-notifications
    +--ro streams
    |  +--ro stream* [name]
    |     +--ro name                        stream
```

```
|     +--ro description                string
|     +--ro replay-support?            empty {replay}?
|     +--ro replay-log-creation-time?  yang:date-and-time {replay}?
|     +--ro replay-log-aged-time?      yang:date-and-time {replay}?
+--rw filters
|  +--rw stream-filter* [identifier]
|     +--rw identifier        filter-id
|     +--rw (filter-spec)?
|        +--:(subtree-filter)
|        |  +--rw subtree-filter?              {subtree}?
|        +--:(xpath-filter)
|           +--rw xpath-filter?     yang:xpath1.0 {xpath}?
+--rw subscription-config {configured}?
|  +--rw subscription* [identifier]
|     +--rw identifier           subscription-id
|     +--rw encoding             encoding
|     +--rw (target)
|     |  +--:(stream)
|     |     +--rw (stream-filter)?
|     |     |  +--:(by-reference)
|     |     |  |  +--rw stream-filter-ref    stream-filter-ref
|     |     |  +--:(within-subscription)
|     |     |     +--rw (filter-spec)?
|     |     |        +--:(subtree-filter)
|     |     |        |  +--rw subtree-filter?            {subtree}?
|     |     |        +--:(xpath-filter)
|     |     |           +--rw xpath-filter?   yang:xpath1.0 {xpath}?
|     |     +--rw stream               stream
|     |     +--rw replay-start-time?   yang:date-and-time {replay}?
|     +--rw stop-time?                 yang:date-and-time
|     +--rw receivers
|     |  +--rw receiver* [address port]
|     |     +--rw address     inet:host
|     |     +--rw port        inet:port-number
|     |     +--rw protocol    transport
|     |     +--rw status?     enumeration
|     +--rw (notification-message-origin)?
|        +--:(interface-originated)
|        |  +--rw source-interface?    if:interface-ref
|        +--:(address-originated)
|           +--rw source-vrf?          string
|           +--rw source-address?      inet:ip-address-no-zone
+--ro subscriptions
   +--ro subscription* [identifier]
      +--ro identifier                 subscription-id
      +--ro configured-subscription?   empty {configured}?
      +--ro encoding                   encoding
      +--ro (target)
```

```
          |  +--:(stream)
          |     +--ro (stream-filter)?
          |     |  +--:(by-reference)
          |     |  |  +--ro stream-filter-ref          stream-filter-ref
          |     |  +--:(within-subscription)
          |     |     +--ro (filter-spec)?
          |     |        +--:(subtree-filter)
          |     |        |  +--ro subtree-filter?             {subtree}?
          |     |        +--:(xpath-filter)
          |     |           +--ro 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 (notification-message-origin)?
          |  +--:(interface-originated)
          |  |  +--ro source-interface?          if:interface-ref
          |  +--:(address-originated)
          |     +--ro source-vrf?                string
          |     +--ro source-address?            inet:ip-address-no-zone
          +--ro receivers
             +--ro receiver* [address port]
                +--ro address                   inet:host
                +--ro port                      inet:port-number
                +--ro protocol                  transport
                +--ro pushed-notifications?     yang:counter64
                +--ro excluded-notifications?   yang:counter64
                +--ro status                    enumeration

  rpcs:
    +---x establish-subscription
    |  +---w input
    |  |  +---w encoding?            encoding
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |     +---w (stream-filter)?
    |  |  |     |  +--:(by-reference)
    |  |  |     |  |  +---w stream-filter-ref    stream-filter-ref
    |  |  |     |  +--:(within-subscription)
    |  |  |     |     +---w (filter-spec)?
    |  |  |     |        +--:(subtree-filter)
    |  |  |     |        |  +---w subtree-filter?             {subtree}?
    |  |  |     |        +--:(xpath-filter)
    |  |  |     |           +---w xpath-filter?   yang:xpath1.0 {xpath}?
    |  |  |     +---w stream               stream
    |  |  |     +---w replay-start-time?   yang:date-and-time {replay}?
    |  |  +---w stop-time?           yang:date-and-time
    |  +--ro output
    |     +--ro subscription-result       subscription-result
```

```
    |     +--ro (result)?
    |        +--:(no-success)
    |        |  +--ro filter-failure?           string
    |        |  +--ro replay-start-time-hint?   yang:date-and-time
    |        +--:(success)
    |           +--ro identifier               subscription-id
    +---x modify-subscription
    |  +---w input
    |  |  +---w identifier?          subscription-id
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |     +---w (stream-filter)?
    |  |  |        +--:(by-reference)
    |  |  |        |  +---w stream-filter-ref    stream-filter-ref
    |  |  |        +--:(within-subscription)
    |  |  |           +---w (filter-spec)?
    |  |  |              +--:(subtree-filter)
    |  |  |              |  +---w subtree-filter?            {subtree}?
    |  |  |              +--:(xpath-filter)
    |  |  |                 +---w xpath-filter?   yang:xpath1.0 {xpath}?
    |  |  +---w stop-time?           yang:date-and-time
    |  +--ro output
    |     +--ro subscription-result    subscription-result
    |     +--ro (result)?
    |        +--:(no-success)
    |           +--ro filter-failure?        string
    +---x delete-subscription
    |  +---w input
    |  |  +---w identifier    subscription-id
    |  +--ro output
    |     +--ro subscription-result    subscription-result
    +---x kill-subscription
       +---w input
       |  +---w identifier    subscription-id
       +--ro output
          +--ro subscription-result    subscription-result

  notifications:
    +---n replay-completed {replay}?
    |  +--ro identifier    subscription-id
    +---n subscription-completed
    |  +--ro identifier    subscription-id
    +---n subscription-started {configured}?
    |  +--ro identifier           subscription-id
    |  +--ro encoding             encoding
    |  +--ro (target)
    |  |  +--:(stream)
    |  |     +--ro (stream-filter)?
```

```
   |  |     |  +--:(by-reference)
   |  |     |  |  +--ro stream-filter-ref    stream-filter-ref
   |  |     |  +--:(within-subscription)
   |  |     |     +--ro (filter-spec)?
   |  |     |        +--:(subtree-filter)
   |  |     |        |  +--ro subtree-filter?               {subtree}?
   |  |     |        +--:(xpath-filter)
   |  |     |           +--ro 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
   +---n subscription-resumed
   |  +--ro identifier    subscription-id
   +---n subscription-modified {configured}?
   |  +--ro identifier           subscription-id
   |  +--ro encoding             encoding
   |  +--ro (target)
   |  |  +--:(stream)
   |  |     +--ro (stream-filter)?
   |  |     |  +--:(by-reference)
   |  |     |  |  +--ro stream-filter-ref    stream-filter-ref
   |  |     |  +--:(within-subscription)
   |  |     |     +--ro (filter-spec)?
   |  |     |        +--:(subtree-filter)
   |  |     |        |  +--ro subtree-filter?               {subtree}?
   |  |     |        +--:(xpath-filter)
   |  |     |           +--ro 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
   +---n subscription-terminated
   |  +--ro identifier        subscription-id
   |  +--ro error-id          subscription-errors
   |  +--ro filter-failure?   string
   +---n subscription-suspended
      +--ro identifier        subscription-id
      +--ro error-id          subscription-errors
      +--ro filter-failure?   string
```

The top-level decompositions of data model are as follows:

- "Streams" contains a list of event streams that are supported by the publisher and against which subscription is allowed.

- "Filters" contains a configurable list of filters that can be applied to a subscription. This allows users to reference an existing filter definition as an alternative to defining a filter inline for each subscription.

- "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are a superset of the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions MUST also specify intended receivers and MAY specify the push source from which to send the stream of notification messages.

- "Subscriptions" contains 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.

The data model also contains a number of YANG Notifications that allow a publisher to signal information about a subscription. Finally, the data model contains a number of RPC definitions that are used to manage dynamic subscriptions.

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 targets beyond event streams.

4.1. Establishing a Subscription

The "establish-subscription" operation allows a subscriber to request the creation of a subscription via RPC. Multiple establish subscription RPC requests can be made within the same transport session.

The input parameters of the operation are:

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

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

- The desired encoding for the notification message.

- An optional stop time for the subscription.

- An optional start time which indicates that this subscription is requesting a replay of previously generated information from the event stream.

If the publisher cannot satisfy the "establish-subscription" request, it sends a negative "subscription-result" element. If the subscriber

has no authorization to establish the subscription, the "subscription-result" indicates an authorization error. Optionally, the "subscription-result" MAY include one or more hints on alternative input parameters and value which would have resulted in an accepted subscription.

Subscription requests MUST fail if a filter with invalid syntax is provided or if a non-existent stream is referenced.

4.1.1. Replay Subscription

Replay provides the ability to establish an subscription which is also capable of passing along recently generated event records. In other words, as the subscription initializes itself, it sends any previously generated content from within target event stream which meet 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 only viable for dynamic subscriptions. Replay is an optional feature. Replay is dependent on an event stream supporting some form of logging, although it 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.

An end time MAY be specified using the optional stop-time parameter, which only in the case of replay MAY also be earlier than the current time. If no stop-time is present, notification messages will continue to be sent until the subscription is terminated. 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.

Not all streams will support replay. Those that do MUST include they do via the "replay-support" object. In addition, a event stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given replay request. Subscribers MAY do a get on "replay-log-creation-time" and "replay-log-aged-time" to assess the availability of replay. The actual size of the replay log at any given time is a publisher

specific matter. Control parameters for this aspect of the feature are outside the scope of this document.

4.2. Modifying a Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription previously established on that transport session. Subscriptions created by configuration operations cannot be modified via this RPC. Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the requested modifications, it replies that this change has been made, then immediately starts sending event records based on the new terms. If the publisher rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of a such a rejected modification MAY include one or more hints on alternative input parameters and value which would have resulted in a successfully modified subscription.

Dynamic subscriptions established via RPC can only be modified via RPC using the same transport session used to establish that subscription.

4.3. 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 this successful reply has been sent, the publisher MUST NOT send any more notification messages for this subscription. If the publisher rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever.

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

4.4. Killing a Subscription

The "kill-subscription" operation permits an operator to end a dynamic subscription which is not associated 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 the yang tree.

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.

5. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface. Such a subscription MAY push notification messages to more than one receiver. The publisher does not provide information about receivers are provided no information about other receivers. Supporting configured subscriptions is optional and advertised using the "configured" feature.

Configured subscriptions persist across reboots, and persist even when transport is unavailable.

Configured subscriptions can be modified by any configuration client with write permissions for the configuration of the subscription. Subscriptions can be modified or terminated via the configuration interface at any point of their lifetime.

In addition to subscription parameters that apply to dynamic subscriptions, the following additional parameters apply to configured subscriptions:

- One or more receiver IP addresses (and corresponding ports) intended as the destination for notification messages for each subscription. In addition, the transport for each destination MAY be defined.

- Optional parameters to identify an egress interface, a host IP address, a VRF, or an IP address plus VRF out of which notification messages should 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.

5.1. Creating a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPCs and configuration operations for subscription creation. Firstly, configuration operations install a subscription without question, while RPCs are designed to the support negotiation and rejection of requests. Secondly, while 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, as successful receipt of the subscription start change notification by a particular receiver indicated, notification messages may then be pushed.

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 is to allows 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.

5.2. Modifying a Configured Subscription

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

Immediately after a subscription is successfully modified, the publisher sends to the existing receivers a state change notification stating the subscription has been modified (i.e., subscription-modified).

If the modification involved adding and/or removing receivers, those modified receivers are sent state change notifications, indicating they have been added (i.e, subscription-started to a specific receiver) or removed (i.e., subscription-terminated to a specific receiver.)

5.3. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example, in RESTCONF:

```
DELETE /subscription-config/subscription=1922 HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Sun, 24 Jul 2016 11:23:40 GMT
Server: example-server
```

Figure 3: Deleting a configured subscription

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

6. Deleting a Configured Subscription

Configured 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 the existing receivers a state change notification stating the subscription has been terminated (i.e., subscription-terminated).

7. Asynchronous Subscribed Event Delivery

Once a subscription has been set up, the publisher streams subscribed 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 specified connections.

A notification message is sent to a receiver when something of interest occurs which is able to traverse all specified filtering and access control 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 4: 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.

8. 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 definition of subscription state notifications is distinct from other notification messages by making use of a YANG extension tagging them as subscription state notification.

Subscription state notifications include indications that a replay of event records has been completed, that a subscription is done because an end time has been reached, and that a subscription has started, been modified, been terminated, or been suspended. They are described in the following subsections.

8.1. subscription-started

This indicates that a configured subscription has started and data updates are beginning to be sent. This state change notification includes 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 for RPC-based subscriptions, no "subscription-started" notifications are sent.

8.2. subscription-modified

This indicates that a configured subscription has been modified successfully. This state change notification includes 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 for RPC-based subscriptions, no "subscription-modified" state change notifications are sent.

8.3. subscription-terminated

This indicates that a subscription has been terminated by the publisher. The state change notification includes the reason for the termination. The publisher MAY decide to terminate a subscription when it is running out of resources for serving it, an internal error occurs, etc. Publisher-driven terminations are notified to all receivers. Northbound systems MAY also terminate configured subscriptions using configuration operations.

Subscribers can terminate via RPC 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 there MUST be this state change notification that the subscription has been ended.

8.4. subscription-suspended

This indicates that a publisher has suspended a subscription. The state change notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further subscribed event records will be sent until the subscription resumes. Suspensions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

8.5. subscription-resumed

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

notifications go to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

8.6. subscription-completed

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

8.7. replay-completed

This 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 event records will be sent in sequence as they arise naturally within the system.

9. Administrative Functions

9.1. Subscription Monitoring

Container "subscriptions" in the YANG module below 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 status of subscriptions to be monitored.

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

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.

9.2. 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, 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.

9.3. Event Stream Discovery

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. A client can retrieve this list like any other YANG-defined data.

10. Data Model

```
<CODE BEGINS> file "ietf-subscribed-notifications.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;
  }

  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 2017-10-27 {
    description
      "Initial version";
    reference
      "draft-ietf-netconf-subscribed-notifications-06";
  }

  /*
   * 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.";
  }

  /*
   * EXTENSIONS
   */

  extension subscription-state-notif {
    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 subscription results */
  identity subscription-result {
    description
      "Base identity for RPC responses and State Change Notifications
      providing information on the creation, modification, deletion of
       subscriptions.";
  }

  identity ok {
    base subscription-result;
    description
      "OK - RPC was successful and was performed as requested.";
  }
```

```
  identity error {
    base subscription-result;
    description
      "Problem with subscription.  Base identity for error return
      codes for RPCs and State Change Notifications.";
  }

  /* Identities for subscription errors */

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

  identity stream-unavailable {
    base error;
    description
     "Stream does not exist or is not available to the receiver.";
  }

  identity encoding-unavailable {
    base error;
    description
     "Encoding not supported";
  }

  identity replay-unsupported {
    base error;
    description
     "Replay cannot be performed for this subscription. The publisher
     does not provide the requested historic information via replay.";
  }

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

  identity filter-unavailable {
    base error;
    description
     "Referenced filter does not exist";
```

```
  }

  identity filter-type-unsupported {
    base error;
    description
     "Publisher does not support filters constructed using this
     filtering technology syntax.";
  }

  identity filter-unsupported {
    base error;
    description
     "Failure can be from a syntax error, or a syntax too complex to be
     processed by the platform. The supplemental info should include
     the invalid part of the filter.";
  }

  identity namespace-unavailable {
    base error;
    description
     "Referenced namespace doesn't exist or is unavailable
      to the receiver.";
  }

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

  identity insufficient-resources {
    base error;
    description
      "The publisher has insufficient resources to support the
       subscription as requested by an RPC.";
  }

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

  identity error-no-such-option {
```

```
    base error;
    description
      "A requested parameter setting is not supported.";
  }

  /* 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 subscription-result {
    type identityref {
      base subscription-result;
    }
    description
      "The result of a subscription operation";
  }

  typedef subscription-errors {
    type identityref {
      base error;
    }
    description
      "The reason for the failure of an RPC request or the sending
       of a subscription suspension or termination state change
       notification";
  }

  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 {
    type string;
    description
      "Specifies 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 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 subtree-filter {
        if-feature "subtree";
        description
          "Event stream evaluation criteria encoded in a syntax of a
           supported type of an RFC 6241, Section 6 filter.  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
           stream filter matches an event record from the stream, the
           event record should be included in a notification message
```

```
           to the receiver(s).";
      }
      leaf xpath-filter {
        if-feature "xpath";
        type yang:xpath1.0;
        description
          "Event stream evaluation criteria encoded in a syntax of xpath
           1.0 and applied against an event stream. The result of
           applying XPath expression is converted to a boolean value
           using the standard XPath 1.0 rules.  If the boolean value is
           'true', the stream filter matches an event record within the
           stream, and the notification message should be sent to the
           receiver(s).";
      }
    }
  }

  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 {
    description
      "This grouping describes information concerning a subscription.";
    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 must be set just by RPC.";
        leaf stream {
          type stream;
          mandatory true;
          description
            "Indicates a stream of event records against which to apply
            a stream filter.";
        }
        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.";
        }
      }
    }
  }

  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 the push source is out of an interface on the
           Publisher established via static configuration.";
        leaf source-interface {
          type if:interface-ref;
          description
            "References the interface for notification messages.";
        }
      }
      case address-originated {
        description
          "When the push source is out of an IP address on the
           Publisher established via static configuration.";
        leaf source-vrf {
          type string;
          description
            "Network instance name for the VRF.  This could also have
            been a leafref to draft-ietf-rtgwg-ni-model, but that model
            is not complete, and might not be implemented on a box.";
        }
        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.";
        }
        leaf protocol {
          type transport;
          mandatory true;
          description
            "This leaf specifies the transport protocol used
             to deliver messages destined for the receiver.  Each
             protocol may use the address and port information
             differently as applicable.";
        }
      }
    }
  }

  grouping error-identifier {
    description
      "A code passed back within an RPC response to describe why the RFC
       has failed, or within a state change notification to describe why
       the change has occurred.";
    leaf error-id {
      type subscription-errors;
      mandatory true;
      description
```

```
        "Identifies the subscription error condition.";
    }
  }

  grouping hints {
    description
      "Objects passed back within an RPC response to describe why the
       RFC has failed, or within a state change notification to
       describe why the change has occurred.";
    leaf filter-failure {
      type string;
      description
        "Information describing where and/or why a provided filter was
        unsupportable for a subscription.";
    }
  }

  grouping subscription-response-with-hints {
    description
      "Defines the output for the establish-subscription and
       modify-subscription RPCs.";
    leaf subscription-result {
      type subscription-result;
      mandatory true;
      description
        "Indicates whether subscription is operational, or if a problem
         was encountered.";
    }
    choice result {
      description
        "Depending on the subscription result, different data is
         returned.";
      case no-success {
        description
          "This case applies when a subscription request was not
           successful and no subscription was created (or modified) as a
           result.  In this case, information MAY be returned that
           indicates suggested parameter settings that would have a
           high likelihood of succeeding in a subsequent establish-
           subscription or modify-subscription request.";
        uses hints;
      }
    }
  }

  /*
   * 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 (as indicated by
       subscription-result) is returned, the subscription is not
       created.  In that case, the RPC reply MAY include suggested
       parameter settings that would have a higher likelihood of
       succeeding in a subsequent establish-subscription request.";
    input {
      uses subscription-policy {
        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.";
        }
      }
    }
    output {
      uses subscription-response-with-hints  {
        augment "result" {
          description
            "Allows information to be passed back as part of a
             successful subscription establishment.";
          case success {
            description
              "This case is used when the subscription request was
               successful.";
            leaf identifier {
              type subscription-id;
              mandatory true;
              description
                "Identifier used for this subscription.";
            }
          }
        }
        augment "result/no-success" {
          description
            "Contains establish RPC specific objects which can be
             returned as hints for future attempts.";
          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 an error
       is returned (as indicated by subscription-result), the
       subscription is not modified and the original subscription
       parameters remain in effect.  In that case, the rpc error
       response MAY include suggested parameter hints that would have
       a high likelihood of succeeding in a subsequent
       modify-subscription request.";
    input {
      leaf identifier {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-modifiable;
    }
    output {
      uses subscription-response-with-hints;
    }
  }

  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.";
      }
```

```
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription has been deleted, or if a
          problem was encountered.";
      }
    }
  }

  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.";
      }
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription has been killed, or if a
           problem was encountered.";
      }
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification replay-completed {
    sn:subscription-state-notif;
    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-notif;
    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-notif;
    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.";
      }
    }
  }
```

```
  notification subscription-resumed {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription that had
       previously been suspended has resumed. Notifications will once
       again be sent.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification subscription-modified {
    sn:subscription-state-notif;
    if-feature "configured";
    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;
  }

  notification subscription-terminated {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription has been
       terminated.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses error-identifier;
    uses hints;
  }

  notification subscription-suspended {
```

```
    sn:subscription-state-notif;
    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.";
    }
    uses error-identifier;
    uses hints;
  }

  /*
   * 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 stream;
        description
          "A handle for 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 subscription-config {
    if-feature "configured";
    description
```

```
      "Contains the list of subscriptions that are configured,
       as opposed to established via RPC or other means.";
    list subscription {
      key "identifier";
      description
        "The identity and specific parameters of a subscription.";
      leaf identifier {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy;
      uses receiver-info {
        augment receivers/receiver {
          description
            "include operational data for receivers.";
          leaf status {
            type enumeration {
              enum connecting {
                value 5;
                description
                  "A subscription has been configured, and a
                  subscription-started state change notification should
                  be sent as quickly as possible.";
              }
              enum suspended {
                value 3;
              description
                "The status is suspended, meaning that the publisher is
                currently will not provide notification messages for
                the subscription until some status change.";
              }
            }
            default "connecting";
            description
              "Allows state initialization of a particular receiver.";
          }
        }
      }
      uses notification-origin-info;
    }
  }
  container subscriptions {
    config false;
    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 parameterst 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 {
       if-feature "configured";
       type empty;
       description
         "The presence of this leaf indicates that the subscription
          originated from configuration, not through a control channel
          or RPC.";
      }
      uses subscription-policy;
      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;
            description
              "Operational data which provides the number of update
               notification messages pushed to a receiver.";
            }
          leaf excluded-notifications {
            type yang:counter64;
            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 status {
            type enumeration {
              enum active {
                value 1;
                description
                  "Connection is active and healthy.";
```

```
              }
              enum concluded {
                value 2;
                description
                  "A subscription is inactive as it has hit a stop time,
                  but not yet been removed.";
              }
              enum suspended {
                value 3;
                description
                  "The status is suspended, meaning that the publisher
                  is currently unable to provide notification messages
                  for the subscription.";
              }
              enum in-error {
                value 4;
                description
                  "The status is in error or degraded, meaning that a
                  subscription is unsupportable with its current
                  parameters.";
              }
              enum connecting {
                value 5;
                description
                  "A subscription has been configured, but a
                  subscription-started state change notification has not
                  yet been succesfully received.";
              }
            }
            mandatory true;
            description
              "Specifies the status 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.";
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

11. Considerations

11.1. 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 SHOULD be used for subscriptions which will have subscription identifiers provided from outside the publisher, and upper half for subscription identifiers assigned by the publisher.

No state change notification or nor subscribed event records within notification messages may be sent before the transport layer, including any requried 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.

11.2. 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 status 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 user 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 user 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. Subscribers 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-user 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.

12. 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.

13. References

13.1. Normative References

[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>.

[RFC6536bis] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-01 (work in progress), September 2017.

[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>.

13.2. 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 2016, <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", October 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>.

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

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

[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)

v06 - v07

- Clarification on state machine for configured susbcriptions.

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 susbcription 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