

NETCONF  
Internet-Draft  
Intended status: Standards Track  
Expires: March 15, 2017

A. Clemm  
A. Gonzalez Prieto  
E. Voit  
E. Nilsen-Nygaard  
A. Tripathy  
Cisco Systems  
S. Chisholm  
Ciena  
H. Trevino  
Cisco Systems  
September 11, 2016

Subscribing to Event Notifications  
draft-ietf-netconf-rfc5277bis-00

Abstract

This document defines capabilities and operations for providing asynchronous message notification delivery for notifications, such as those defined using YANG. Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF and RESTCONF. The capabilities and operations defined in this document along with their mapping onto NETCONF transport are intended to replace [RFC 5277](#).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on March 15, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

Internet-Draft

Event Notifications

September 2016

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">1.1.</a>	Motivation . . . . .	<a href="#">3</a>
<a href="#">1.2.</a>	Terminology . . . . .	<a href="#">4</a>
<a href="#">1.3.</a>	Solution Overview . . . . .	<a href="#">5</a>
<a href="#">2.</a>	Solution . . . . .	<a href="#">6</a>
<a href="#">2.1.</a>	Event Streams . . . . .	<a href="#">6</a>
<a href="#">2.2.</a>	Event Stream Discovery . . . . .	<a href="#">7</a>
<a href="#">2.3.</a>	Filters . . . . .	<a href="#">7</a>
<a href="#">2.4.</a>	Subscription State Model at the Event Server . . . . .	<a href="#">7</a>
<a href="#">3.</a>	Data Model Trees for Event Notifications . . . . .	<a href="#">8</a>
<a href="#">4.</a>	Dynamic Subscriptions . . . . .	<a href="#">12</a>
<a href="#">4.1.</a>	Establishing a Subscription . . . . .	<a href="#">13</a>
<a href="#">4.2.</a>	Modifying a Subscription . . . . .	<a href="#">13</a>
<a href="#">4.3.</a>	Deleting a Subscription . . . . .	<a href="#">14</a>
<a href="#">5.</a>	Configured Subscriptions . . . . .	<a href="#">15</a>
<a href="#">5.1.</a>	Creating a Configured Subscription . . . . .	<a href="#">15</a>
<a href="#">5.2.</a>	Establishing a Configured Subscription . . . . .	<a href="#">15</a>
<a href="#">5.3.</a>	Modifying a Configured Subscription . . . . .	<a href="#">17</a>
<a href="#">5.4.</a>	Deleting a Configured Subscription . . . . .	<a href="#">17</a>
<a href="#">6.</a>	Event (Data Plane) Notifications . . . . .	<a href="#">18</a>
<a href="#">7.</a>	Control Plane Notifications . . . . .	<a href="#">20</a>
<a href="#">7.1.</a>	replayComplete . . . . .	<a href="#">20</a>
<a href="#">7.2.</a>	notificationComplete . . . . .	<a href="#">20</a>
<a href="#">7.3.</a>	subscription-started . . . . .	<a href="#">20</a>
<a href="#">7.4.</a>	subscription-modified . . . . .	<a href="#">21</a>
<a href="#">7.5.</a>	subscription-terminated . . . . .	<a href="#">21</a>
<a href="#">7.6.</a>	subscription-suspended . . . . .	<a href="#">21</a>
<a href="#">7.7.</a>	subscription-resumed . . . . .	<a href="#">21</a>
<a href="#">8.</a>	Data Model for Event Notifications . . . . .	<a href="#">21</a>
<a href="#">9.</a>	Backwards Compatibility . . . . .	<a href="#">39</a>
<a href="#">10.</a>	Security Considerations . . . . .	<a href="#">40</a>

<a href="#">11.</a>	Issues that are currently being worked and resolved . . . . .	<a href="#">41</a>
<a href="#">11.1.</a>	Unresolved and yet-to-be addressed issues . . . . .	<a href="#">41</a>
<a href="#">11.2.</a>	Agreement in principal . . . . .	<a href="#">41</a>
<a href="#">11.3.</a>	Resolved Issues . . . . .	<a href="#">42</a>
<a href="#">12.</a>	Acknowledgments . . . . .	<a href="#">42</a>

<a href="#">13.</a>	References . . . . .	<a href="#">42</a>
<a href="#">13.1.</a>	Normative References . . . . .	<a href="#">42</a>
<a href="#">13.2.</a>	Informative References . . . . .	<a href="#">43</a>
	Authors' Addresses . . . . .	<a href="#">43</a>

## [1.](#) Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service in a protocol-agnostic manner. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications including those necessary to establish, monitor, and support subscriptions to notification delivery.

Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF [[RFC6241](#)] (defined in [[I-D.ietf-netconf-netconf-event-notif](#)]) and Restconf [[I-D.ietf-netconf-restconf](#)] (defined in [[I-D.ietf-netconf-restconf-notif](#)]). The capabilities and operations defined in this document are intended to replace [RFC 5277](#), along with their mapping onto NETCONF transport.

### [1.1.](#) Motivation

The motivation for this work is to enable the sending of asynchronous notification messages that are consistent with the data model (content) and security model used within a NETCONF implementation.

[[RFC5277](#)] defines a notification mechanism for NETCONF. However, there are various limitations:

- o Each subscription requires a separate NETCONF connection, which is wasteful.
- o The only mechanism to terminate a subscription is terminating the underlying NETCONF connection.

- o No ability to modify subscriptions once they have been created.
- o No ability to notify the receiver of a subscription if the server is dropping events.
- o No mechanism to monitor subscriptions.
- o No alternative mechanism to create subscriptions via RPCs. Thus the lifetime of the subscription is limited by that of the underlying NETCONF session.

- o Predates YANG and defines RPCs, notifications, and data nodes outside of the YANG framework.

The scope of the work aims at meeting the following operational needs:

- o Ability to dynamically or statically subscribe to event notifications available on an event server.
- o Ability to negotiate acceptable dynamic subscription parameters.
- o Ability to filter the subset of notifications to be pushed with stream-specific semantics.
- o Ability for the notification payload to be interpreted independently of the transport protocol. (In other words, the encoded notification fully describes itself.)
- o Mechanism to communicate the notifications.
- o Ability to replay locally logged notifications.

## [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 event server via create, establish, modify, and delete RPC control plane signaling messages.

**Event:** Something that happens 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 notification:** A message sent by a server to a receiver indicating that an event (of interest to the subscriber) has occurred. Events can trigger notifications if an interested party has subscribed to the stream(s) it belongs to.

**Event server:** The server being subscribed to, which serves an event subscription.

**Filter:** Evaluation criteria, which may be applied against a targeted set of objects/events in a subscription. Information traverses the filter only if specified filter criteria are met.

**NACM:** NETCONF Access Control Model.

**OAM:** Operations, Administration, Maintenance.

**Publisher:** An entity responsible for streaming Event Notifications per the terms of a Subscriptions

**RPC:** Remote Procedure Call.

**Stream** (also referred to as "event stream"): A continuous ordered set of events grouped under an explicit criteria.

**Subscriber:** An entity able to request and negotiate a contract for the receipt of event notifications from a NETCONF server.

**Receiver:** A target to which an event server pushes event notifications. In many deployments, the receiver and subscriber will be the same entity.

Subscription: A contract between a subscriber and an event server, stipulating which information the receiver wishes to have pushed from the server without the need for further solicitation.

### 1.3. Solution Overview

This document describes mechanisms for subscribing and receiving event notifications from an event server. This document builds on top of the capabilities defined in [[RFC5277](#)], extending them, and generalizing them to be protocol-agnostic.

The enhancements over [RFC 5277](#) include the ability to terminate subscriptions without terminating the client session, to modify existing subscriptions, and to have multiple subscriptions on a NETCONF session.

These enhancements do not affect existing [RFC 5277](#) clients that do not support these particular subscription requirements.

The solution supports subscribing to event notifications using two mechanisms.

1. Dynamic subscriptions, where a client initiates a subscription negotiation with an event server via RPC. A client initiates a negotiation by issuing a subscription request. If the event

server wants to serve this request, it will accept it, and then start pushing event notifications as negotiated. If the event server does not wish to serve it as requested, it may respond with subscription parameters which it would have accepted.

2. Configured subscriptions, which is an optional mechanism that enables managing subscriptions via a configuration interface so that an event server sends event notifications to a configured receiver(s).

Some key characteristics of configured and dynamic subscriptions include:

- o The lifetime of a dynamic subscription is limited by the lifetime of the subscriber session used to establish it. Typically loss of

the transport session tears down any dependent dynamic subscriptions.

- o The lifetime of a configured subscription is driven by configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persists even when transport is unavailable. This also means configured subscriptions do not support negotiation.
- o Subscriptions can be modified or terminated at any point of their lifetime. configured subscriptions can be modified by any configuration client with write rights on the configuration of the subscription.

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 created via RPC cannot be modified through configuration operations.

The event server may decide to terminate a dynamic subscription at any time. Similarly, it may decide to temporarily suspend the sending of event notifications for either configured or dynamic subscriptions. Such termination or suspension may be driven by the server running out of resources to serve the subscription, or by internal errors on the server.

## [2.](#) Solution

### [2.1.](#) Event Streams

An event stream is a set of events available for subscription from a server. 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.

That said, some event streams will be standardized whereas others may be vendor specific. One standardized event stream is the "NETCONF" notification event stream. The NETCONF event stream contains all NETCONF XML event notifications supported by the NETCONF server, except for those belonging only to streams that explicitly indicate

that they must be excluded from the NETCONF stream, such as notifications that serve OAM and signaling purposes.

The following is a high-level description of the flow of a notification. Note that it does not mandate and/or preclude an implementation. As events are raised, they are assigned to streams. An event may be assigned to multiple streams. The event is distributed to subscribers and receivers based on the current subscriptions and access control. Access control is needed because if any receiver of that subscription does not have permission to receive an event, then it never makes it into a notification, and processing of the event is completed for that subscription.

## [2.2.](#) Event Stream Discovery

A server 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, for example using the <get> operation when using NETCONF.

## [2.3.](#) Filters

An event server implementation SHOULD support the ability to perform filtering of notification records per [RFC 5277](#). (TODO: since 5277 is to be obsoleted, we should describe the filter here.)

## [2.4.](#) Subscription State Model at the Event Server

Below is the state machine of a subscription for the event server. It is important to note that a subscription doesn't exist at the event server 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.

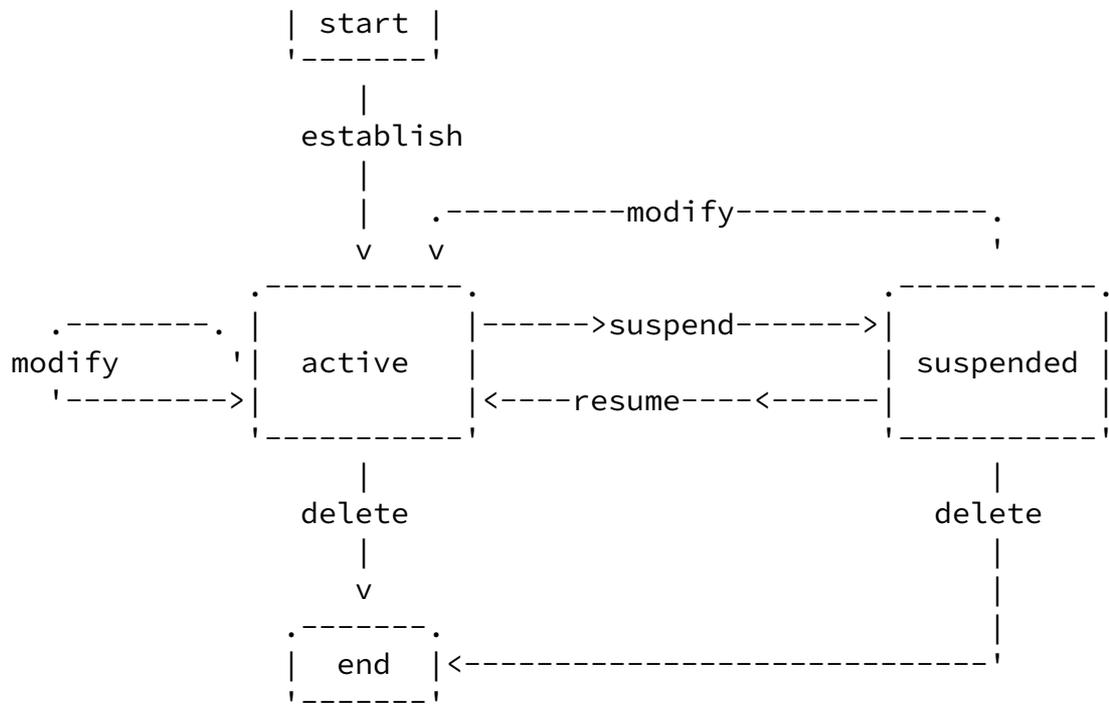


Figure 1: Subscription states at event server

Of interest in this state machine are the following:

- o Successful <establish-subscription> or <modify-subscription> requests put the subscription into an active state.
- o Failed <modify-subscription> requests will leave the subscription in its previous state, with no visible change to any streaming updates.
- o A <delete-subscription> request will delete the entire subscription.

### 3. Data Model Trees for Event Notifications

The YANG data model for event notifications is depicted in this section.

```

module: ietf-event-notifications
  +--ro streams
  | +--ro stream*  notif:stream
  +--rw filters
  | +--rw filter* [filter-id]
  |   +--rw filter-id  filter-id
  |   +--rw (filter-type)?
  |   +--:(rfc5277)
  
```

```

|         +---rw filter
+---rw subscription-config {configured-subscriptions}?
|   +---rw subscription* [subscription-id]
|     +---rw subscription-id      subscription-id
|     +---rw stream?              stream
|     +---rw (filter-type)?
|       | +---:(rfc5277)
|       | | +---rw filter
|       | +---:(by-reference)
|       | | +---rw filter-ref?          filter-ref
+---rw startTime?                yang:date-and-time
+---rw stopTime?                 yang:date-and-time
+---rw encoding?                  encoding
+---rw receivers
|   +---rw receiver* [address]
|     +---rw address              inet:host
|     +---rw port                  inet:port-number
|     +---rw protocol?            transport-protocol
+---rw (push-source)?
|   +---:(interface-originated)
|     | +---rw source-interface?      if:interface-ref
|   +---:(address-originated)
|     +---rw source-vrf?             uint32
|     +---rw source-address          inet:ip-address-no-zone
+---ro subscriptions
|   +---ro subscription* [subscription-id]
|     +---ro subscription-id        subscription-id
|     +---ro configured-subscription? empty {configured-subscriptions}?
|     +---ro subscription-status?   subscription-status
|     +---ro stream?                stream
|     +---ro (filter-type)?
|       | +---:(rfc5277)
|       | | +---ro filter
|       | +---:(by-reference)
|       | | +---ro filter-ref?          filter-ref
+---ro startTime?                 yang:date-and-time
+---ro stopTime?                   yang:date-and-time
+---ro encoding?                    encoding
+---ro receivers
|   +---ro receiver* [address]
|     +---ro address                inet:host
|     +---ro port                    inet:port-number
|     +---ro protocol?               transport-protocol
+---ro (push-source)?
|   +---:(interface-originated)
|     | +---ro source-interface?       if:interface-ref

```

+---:(address-originated)  
+---ro source-vrf? uint32

```
+---ro source-address inet:ip-address-no-zone
rpcs:
+---x establish-subscription
| +---w input
| | +---w stream? stream
| | +---w (filter-type)?
| | | +---:(rfc5277)
| | | | +---w filter
| | | +---:(by-reference)
| | | +---w filter-ref? filter-ref
| | +---w startTime? yang:date-and-time
| | +---w stopTime? yang:date-and-time
| | +---w encoding? encoding
| +---ro output
| +---ro subscription-result subscription-result
| +---ro (result)?
| | +---:(success)
| | | +---ro subscription-id subscription-id
| | +---:(no-success)
| | +---ro stream? stream
| | +---ro (filter-type)?
| | | +---:(rfc5277)
| | | | +---ro filter
| | | +---:(by-reference)
| | | +---ro filter-ref? filter-ref
| | +---ro startTime? yang:date-and-time
| | +---ro stopTime? yang:date-and-time
| | +---ro encoding? encoding
+---x modify-subscription
| +---w input
| | +---w subscription-id? subscription-id
| | +---w stream? stream
| | +---w (filter-type)?
| | | +---:(rfc5277)
| | | | +---w filter
| | | +---:(by-reference)
| | | +---w filter-ref? filter-ref
| | +---w startTime? yang:date-and-time
| | +---w stopTime? yang:date-and-time
```

```

| | +---w encoding?          encoding
| +--ro output
|   +--ro subscription-result  subscription-result
|   +--ro (result)?
|     +--:(success)
|       | +--ro subscription-id      subscription-id
|       +--:(no-success)
|         +--ro stream?              stream
|         +--ro (filter-type)?

```

```

|         | +--:(rfc5277)
|         | | +--ro filter
|         | +--:(by-reference)
|         |   +--ro filter-ref?          filter-ref
|         +--ro startTime?              yang:date-and-time
|         +--ro stopTime?               yang:date-and-time
|         +--ro encoding?                encoding
+---x delete-subscription
  +---w input
  | +---w subscription-id      subscription-id
  +--ro output
    +--ro subscription-result  subscription-result
notifications:
+---n replay-complete
| +--ro subscription-id      subscription-id
+---n notification-complete
| +--ro subscription-id      subscription-id
+---n subscription-started
| +--ro subscription-id      subscription-id
| +--ro stream?              stream
| +--ro (filter-type)?
| | +--:(rfc5277)
| | | +--ro filter
| | +--:(by-reference)
| |   +--ro filter-ref?      filter-ref
| +--ro startTime?          yang:date-and-time
| +--ro stopTime?           yang:date-and-time
| +--ro encoding?           encoding
+---n subscription-suspended
| +--ro subscription-id      subscription-id
| +--ro reason?              subscription-susp-reason
+---n subscription-resumed

```

```

| +--ro subscription-id    subscription-id
+---n subscription-modified
| +--ro subscription-id    subscription-id
| +--ro stream?            stream
| +--ro (filter-type)?
| | +--:(rfc5277)
| | | +--ro filter
| | | +--:(by-reference)
| | | +--ro filter-ref?    filter-ref
| +--ro startTime?        yang:date-and-time
| +--ro stopTime?        yang:date-and-time
| +--ro encoding?        encoding
+---n subscription-terminated
| +--ro subscription-id    subscription-id
| +--ro reason?            subscription-term-reason
+---n added-to-subscription

```

```

| +--ro subscription-id    subscription-id
| +--ro stream?            stream
| +--ro (filter-type)?
| | +--:(rfc5277)
| | | +--ro filter
| | | +--:(by-reference)
| | | +--ro filter-ref?    filter-ref
| +--ro startTime?        yang:date-and-time
| +--ro stopTime?        yang:date-and-time
| +--ro encoding?        encoding
+---n removed-from-subscription
  +--ro subscription-id    subscription-id

```

The data model is structured as follows:

- o "Streams" contains a list of event streams that are supported by the event server and that can be subscribed to.
- o "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.
- o "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are

equivalent to the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions specify intended receivers and the push source from which to send the stream of notification messages.

- o "Subscriptions" contains a list of all subscriptions on an event server, both configured and dynamic. It can be used to retrieve information about the subscriptions which an event server is serving.

The data model also contains a number of notifications that allow an event server to signal to the client 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.

##### [4.1.](#) Establishing a Subscription

This operation is an evolution of the "create-subscription" operation defined in [RFC 5277](#). It allows a subscriber to request the creation of a subscription both via RPC and configuration operations. When invoking the RPC, establish-subscription permits negotiating the subscription terms, changing them dynamically.

The input parameters of the operation are those of create subscription plus:

- o `filter-ref`: filters that have been previously (and separately) configured can be referenced by a subscription. This mechanism enables the reuse of filters.
- o `encoding`: by default, updates are encoded using XML. Other encodings may be supported, such as JSON.

If the event server cannot satisfy the request, it sends a negative <subscription-result> element.

If the client has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. If the request is rejected because the server is not able to serve it, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request when it was processed. However, there is no guarantee that subsequent requests with those parameters for this client or others will be accepted. For instance, consider a subscription from [[I-D.ietf-netconf-yang-push](#)], which augments the establish-subscription with some additional parameters, including "period".

Subscription requests will fail if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

#### [4.2.](#) Modifying a Subscription

This operation permits modifying the terms of a subscription previously established. Subscriptions created by configuration cannot be modified. Dynamic subscriptions can be modified one or multiple times. If the server accepts the request, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the server rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of negative responses to modify-subscription requests are the same as in establish-subscription requests.

Dynamic subscriptions established via RPC can only be modified (or deleted) via RPC using the same session used to establish it. Configured subscriptions cannot be modified (or deleted) using RPCs. Instead, configured subscriptions are modified (or deleted) as part of regular configuration operations. Servers MUST reject any attempts to modify (or delete) configured subscriptions via RPC.

The parameters to modify-subscription are those of establish-subscription plus a mandatory subscription-id.

If the event server can satisfy the request, the server sends a

positive subscription-result. This response is like that to an establish-subscription request without the subscription-id, which would be redundant.

If the event server cannot satisfy the request, the server sends a negative subscription-result. Its contents and semantics are identical to those to an establish-subscription request.

#### [4.3.](#) Deleting a Subscription

This operation permits canceling a subscription previously established. Created subscriptions cannot be explicitly deleted. If the server accepts the request, it immediately stops sending events for the subscription. If the server rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever. A request may be rejected because the provided subscription identifier is incorrect.

Subscriptions created via RPC can only be deleted via RPC using the same session used for establishment. Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Servers MUST reject any RPC attempt to delete configured subscriptions.

The only parameter to delete-subscription is the identifier of the subscription to delete.

If the event server can satisfy the request, the server sends an OK element.

If the event server cannot satisfy the request, the server sends an error-rpc element.

#### [5.](#) Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface.

Configured subscriptions persist across reboots, and persist even when transport is unavailable. This also means configured subscriptions do not support negotiation.

Configured subscriptions can be modified by any configuration client with write rights on the configuration of the subscription. Subscriptions can be modified or terminated at any point of their lifetime.

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

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

- o One or more receiver IP addresses (and corresponding ports) intended as the destination for push updates for each subscription. In addition the transport protocol for each destination may be defined.
- o Optional parameters to identify an egress interface or IP address / VRF where a subscription updates should be pushed from the publisher.

### [5.1.](#) Creating a Configured Subscription

Configured subscriptions cannot be created via configuration operations. New clients should use the mechanisms described in [Section 5.2](#) for establishing configured subscriptions.

### [5.2.](#) Establishing a Configured Subscription

Subscriptions can be established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and configuration operations for subscription establishment. Firstly, configuration operations do not support negotiation while RPCs do. Secondly, while RPCs mandate that the client establishing the subscription is the only receiver of the notifications, configuration operations permit specifying receivers independent of any tracked subscriber. Immediately after a subscription is successfully established, the server sends to the

receivers a control-plane notification stating the subscription has been established (subscription-started).

Because there is no explicit association with an existing transport session, configured configuration operations require additional parameters to indicate the receivers of the notifications and possibly the source of the notifications (i.e., a specific interface or server address).

For example at subscription establishment, a NETCONF client may send:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
      <subscription>
        <subscription-id>
          1922
        </subscription-id>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.4
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 2: Establish configured subscription

if the request is accepted, the server would reply:

Internet-Draft

Event Notifications

September 2016

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

Figure 3: Response to a successful configured subscription establishment

if the request is not accepted because the server cannot serve it, the server may reply:

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

Figure 4: Response to a failed configured subscription establishment

### [5.3.](#) 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 server sends to the existing receivers a control-plane 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 control-plane notifications, indicating they have been added (i.e., `added-to-subscription`, with the same contents as a `modified-subscription`) or removed (i.e., `removed-from-subscription`).

#### 5.4. Deleting a Configured Subscription

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

Clemm, et al.

Expires March 15, 2017

[Page 17]

---

Internet-Draft

Event Notifications

September 2016

```
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 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the event server sends to the receivers a control-plane notification stating the subscription has been terminated (`subscription-terminated`).

#### 6. Event (Data Plane) Notifications

Once a subscription has been set up, the event server sends (asynchronously) the event notifications from the subscribed stream. We refer to these as data plane notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the session used to create or establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to the receiver(s) when an event of interest (i.e., meeting the specified filtering criteria) has occurred. An event notification is a complete and well-formed XML document. Note that `<notification>` is not a Remote Procedure Call (RPC) method but rather the top-level element identifying the one-way message as a notification. Note that event notifications never trigger responses.

The event notification always includes an `<eventTime>` element. It is

the time the event was generated by the event source. This parameter is of type `dateTime` and compliant to [\[RFC3339\]](#). Implementations must support time zones.

The event notification also contains notification-specific tagged content, if any. With the exception of `<eventTime>`, the content of the notification is beyond the scope of this document.

For the encodings other than XML, notifications include an additional XML element so that the notification is a well-formed XML. The element is `<notification-contents-{encoding}>`, E.g., `<notification-contents-json>`. That element contains the notification contents in the desired encoding

The following is an example of an event notification from [\[RFC6020\]](#):

```
notification link-failure {
  description "A link failure has been detected";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

Figure 6: Definition of a data plane notification

```
<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 7: Data plane notification

The equivalent using json encoding would be

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

Figure 8: Data plane notification using JSON encoding

## [7.](#) Control Plane Notifications

In addition to data plane notifications, a server may send control plane notifications to indicate to receivers that an event related to the subscription management has occurred.

Control plane notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to the receiver of a subscription. The definition of control plane notifications is distinct from other notifications by making use of a YANG extension tagging them as control plane notification.

Control plane notifications include indications that a replay of notifications has been completed, that a subscription is done sending notifications 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.

### [7.1.](#) replayComplete

This notification is originally defined in [[RFC5277](#)]. It is sent to indicate that all of the replay notifications have been sent and must not be sent for any other reason.

In the case of a subscription without a stop time, after the <replayComplete> notification has been sent, it can be expected that any notifications generated since the start of the subscription creation will be sent, followed by notifications as they arise naturally within the system.

### [7.2.](#) notificationComplete

This notification is originally defined in [[RFC5277](#)]. It is sent to indicate that a subscription, which includes a stop time, has finished passing events.

### [7.3.](#) subscription-started

This notification indicates that a configured subscription has started and data updates are beginning to be sent. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

### [7.4.](#) subscription-modified

This notification indicates that a configured subscription has been modified successfully. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

### [7.5.](#) subscription-terminated

This notification indicates that a subscription has been terminated. The notification includes the reason for the termination. A

subscription may be terminated by a server or by a client. The server may decide to terminate a subscription when it is running out of resources for serving it, an internal error occurs, etc. Server-driven terminations are notified to all receivers. The management plane can also terminate configured subscriptions using configuration operations.

Clients can terminate via RPC subscriptions established via RPC. In such cases, no subscription-terminated notifications are sent.

#### [7.6.](#) subscription-suspended

This notification indicates that a server has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications 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).

#### [7.7.](#) subscription-resumed

This notification indicates that a previously suspended subscription has been resumed. Data plane notifications generated in the future will be sent after the subscription terms. Resumptions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

### [8.](#) Data Model for Event Notifications

```
<CODE BEGINS>
file "ietf-event-notifications@2016-09-11.yang"
module ietf-event-notifications {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-event-notifications";
```

```
prefix notif-bis;

import ietf-inet-types {
  prefix inet;
}
```

```
import ietf-5277-netmod {
  prefix netmod-notif;
}
import ietf-5277-netconf {
  prefix notif;
}
import ietf-interfaces {
  prefix if;
}

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

  WG Chair: Mahesh Jethanandani
            <mailto:mjethanandani@gmail.com>

  WG Chair: Mehmet Ersue
            <mailto:mehmet.ersue@nokia.com>

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

  Editor: Alberto Gonzalez Prieto
          <mailto:albertgo@cisco.com>

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

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

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

  Editor: Sharon Chisholm
          <mailto:schishol@ciena.com>

  Editor: Hector Trevino
          <mailto:htrevino@cisco.com">
```

description

```

    "This module contains conceptual YANG specifications
    for NETCONF Event Notifications.";

revision 2016-09-11 {
    description
        "Initial version. Model for NETCONF Notifications (bis)";
    reference
        "draft-ietf-netconf-rfc5277bis-00";
}

/*
 * FEATURES
 */

feature json {
    description
        "This feature indicates that JSON encoding of notifications
        is supported.";
}

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

/*
 * IDENTITIES
 */

/* Identities for subscription results */
identity subscription-result {
    description
        "Base identity for RPC responses to requests surrounding
        management (e.g. 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

```

Internet-Draft

Event Notifications

September 2016

```
    "RPC was not successful.
    Base identity for error return codes.";
}

identity error-no-such-subscription {
    base error;
    description
        "A subscription with the requested subscription ID
        does not exist.";
}

identity error-no-such-option {
    base error;
    description
        "A requested parameter setting is not supported.";
}

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

identity error-configured-subscription {
    base error;
    description
        "Cannot apply RPC to a configured subscription, i.e.
        to a subscription that was not established via RPC.";
}

identity error-other {
    base error;
    description
        "An unspecified error has occurred (catch all).";
}

/* Identities for subscription stream status */
identity subscription-stream-status {
    description
        "Base identity for the status of subscriptions and
        datastreams.";
```

```
}  
  
identity active {  
  base subscription-stream-status;  
  description  
    "Status is active and healthy.";
```

```
}  
  
identity inactive {  
  base subscription-stream-status;  
  description  
    "Status is inactive, for example outside the  
    interval between start time and stop time.";  
}  
  
identity suspended {  
  base subscription-stream-status;  
  description  
    "The status is suspended, meaning that the push server  
    is currently unable to provide the negotiated updates  
    for the subscription.";  
}  
  
identity in-error {  
  base subscription-stream-status;  
  description  
    "The status is in error or degraded, meaning that  
    stream and/or subscription is currently unable to provide  
    the negotiated notifications.";  
}  
  
/* Identities for subscription errors */  
identity subscription-errors {  
  description  
    "Base identity for subscription error status.  
    This identity is not to be confused with error return  
    codes for RPCs";  
}  
  
identity internal-error {  
  base subscription-errors;
```

```
    description
      "Subscription failures caused by server internal error.";
  }

  identity no-resources {
    base subscription-errors;
    description
      "Lack of resources, e.g. CPU, memory, bandwidth";
  }

  identity subscription-deleted {
    base subscription-errors;
    description
```

```
    "The subscription was terminated because the subscription
      was deleted.";
  }

  identity other {
    base subscription-errors;
    description
      "Fallback reason - any other reason";
  }

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

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

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

```

/* Identities for transports */
identity transport {
    description
        "An identity that represents a transport protocol for event
        notifications";
}

identity netconf {
    base transport;
    description
        "Netconf notifications as a transport.";
}

/*
 * TYPEDEFS
 */

typedef subscription-id {
    type uint32;
    description

```

```

    "A type for subscription identifiers.";
}

typedef filter-id {
    type uint32;
    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-term-reason {
    type identityref {
        base subscription-errors;

```

```

    }
    description
        "Reason for a server to terminate a subscription.";
}

typedef subscription-susp-reason {
    type identityref {
        base subscription-errors;
    }
    description
        "Reason for a server to suspend a subscription.";
}

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

typedef subscription-status {
    type identityref {
        base subscription-stream-status;
    }
    description
        "Specifies the status of a subscription or datastream.";
}

```

```

}

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

typedef push-source {
    type enumeration {
        enum "interface-originated" {

```

```

        description
            "Notifications will be sent from a specific interface on a
            NETCONF server";
    }
    enum "address-originated" {
        description
            "Notifications will be sent from a specific address on a
            NETCONF server";
    }
}
description
    "Specifies from where notifications will be sourced when
    being sent by the NETCONF server.";
}

typedef filter-ref {
    type leafref {
        path "/notif-bis:filters/notif-bis:filter/notif-bis:filter-id";
    }
    description
        "This type is used to reference a filter.";
}

/*
 * GROUPINGS
 */

grouping subscription-info {
    description
        "This grouping describes basic information concerning a
        subscription.";
    uses notif:subscription-info-5277 {
        augment "filter-type" {
            description

```

```

        "Post-5277 subscriptions allow references to existing
        filters";
    case by-reference {
        description
            "Incorporate a filter that has been configured
            separately.";
        leaf filter-ref {

```

```

        type filter-ref;
        description
            "References filter which is associated with the
            subscription.";
    }
}
}
}
leaf encoding {
    type encoding;
    default "encode-xml";
    description
        "The type of encoding for the subscribed data.
        Default is XML";
}
}
}

grouping push-source-info {
    description
        "Defines the sender source from which notifications
        for a configured subscription are sent.";
    choice push-source {
        description
            "Identifies the egress interface on the Publisher from
            which notifications will or are being 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 notifications.";
            }
        }
        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 uint32 {

```

```

        range "16..1048574";
    }
    description
        "Label of the vrf.";
    }
    leaf source-address {
        type inet:ip-address-no-zone;
        mandatory true;
        description
            "The source address for the notifications.";
    }
    }
}

grouping receiver-info {
    description
        "Defines where and how to deliver notifications for a
        configured subscription. This includes
        specifying the receiver, as well as defining
        any network and transport aspects when sending of
        notifications occurs outside of Netconf.";
    container receivers {
        description
            "Set of receivers in a subscription.";
        list receiver {
            key "address";
            min-elements 1;
            description
                "A single host or multipoint address intended as a target
                for the notifications for a subscription.";
            leaf address {
                type inet:host;
                mandatory true;
                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;
                mandatory true;
                description
                    "This leaf specifies the port number to use for messages
                    destined for a receiver.";
            }
            leaf protocol {

```

Internet-Draft

Event Notifications

September 2016

```
        type transport-protocol;
        default "netconf";
        description
            "This leaf specifies the transport protocol used
            to deliver messages destined for the receiver.";
    }
}
}
}

grouping subscription-response {
    description
        "Defines the output to the rpc's establish-subscription
        and modify-subscription.";
    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 success {
            description
                "This case is used when the subscription request
                was successful and a subscription was created/modified
                as a result";
            leaf subscription-id {
                type subscription-id;
                mandatory true;
                description
                    "Identifier used for this subscription.";
            }
        }
        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 subscription-info;
```

```
    }
  }
}

/*
 * 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 by virtue of a delete-subscription request.
    In case an error (as indicated by subscription-result)
    is returned, the subscription is
    not created. In that case, the RPC output
    MAY include suggested parameter settings
    that would have a high likelihood of succeeding in a
    subsequent create-subscription request.";
  input {
    uses subscription-info;
  }
  output {
    uses subscription-response;
  }
}

rpc modify-subscription {
  description
    "This RPC allows a subscriber to modify a subscription
    that was previously created using create-subscription.
    If successful, the subscription
    remains in effect for the duration of the subscriber's
```

association with the publisher, or until the subscription is terminated by virtue of a delete-subscription request. 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 settings that would have a high likelihood of succeeding in a subsequent modify-subscription request.";

```
input {
  leaf subscription-id {
    type subscription-id;
```

```
    description
      "Identifier to use for this subscription.";
  }
  uses subscription-info;
}
output {
  uses subscription-response;
}
}

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

```

        "Indicates whether subscription is operational,
        or if a problem was encountered.";
    }
}

/*
 * NOTIFICATIONS
 */

notification replay-complete {
    netmod-notif:control-plane-notif;
    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 subscription-id {
        type subscription-id;

```

```

        mandatory true;
        description
            "This references the affected subscription.";
    }
}

notification notification-complete {
    netmod-notif:control-plane-notif;
    description
        "This notification is sent to indicate that a
        subscription, which includes a stop time, has
        finished passing events.";
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
}

notification subscription-started {
    netmod-notif:control-plane-notif;

```

```

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 subscription-id {
  type subscription-id;
  mandatory true;
  description
    "This references the affected subscription.";
}
uses subscription-info;
}

```

```

notification subscription-suspended {
  netmod-notif:control-plane-notif;
  description
    "This notification indicates that a suspension of the
    subscription by the server has occurred. No further
    notifications will be sent until subscription
    resumes.
    This notification shall only be sent to receivers
    of a subscription; it does not constitute a general-purpose
    notification.";
  leaf subscription-id {

```

```

  type subscription-id;
  mandatory true;
  description
    "This references the affected subscription.";
}
leaf reason {
  type subscription-susp-reason;
  description
    "Provides a reason for why the subscription was
    suspended.";
}
}

notification subscription-resumed {
  netmod-notif:control-plane-notif;

```

```

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

```

```

notification subscription-modified {
  netmod-notif:control-plane-notif;
  description
    "This notification indicates that a subscription has
    been modified. Notifications sent from this point
    on will conform to the modified terms of the
    subscription.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses subscription-info;
}

```

```

notification subscription-terminated {
  netmod-notif:control-plane-notif;
  description
    "This notification indicates that a subscription has been
    terminated.";
}

```

```

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

```

```

        description
            "Provides a reason for why the subscription was
            terminated.";
    }
}

notification added-to-subscription {
    netmod-notif:control-plane-notif;
    description
        "This notification is sent to a receiver when it has been
        added to an existing subscription.
        Note that if the receiver is added when the subscription
        is created, it will receive a subscription-started
        notification and no added-to-subscription.";
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
    uses subscription-info;
}

notification removed-from-subscription {
    netmod-notif:control-plane-notif;
    description
        "This notification is sent to a receiver when it has been
        removed from an existing subscription.
        Note that if the subscription is terminated, the receiver
        will receive a subscription-terminated notification
        and no removed-from-subscription.";
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
}
}

/*

```

```

*/
container streams {
    config false;
    description
        "This container contains a leaf list of built-in
        streams that are provided by the system.";
    leaf-list stream {
        type notif:stream;
        description
            "Identifies the built-in streams that are supported by the
            system. Built-in streams are associated with their own
            identities, each of which carries a special semantics.
            In case configurable custom streams are supported,
            as indicated by the custom-stream identity, the configuration
            of those custom streams is provided separately.";
    }
}
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 filter {
        key "filter-id";
        description
            "A list of configurable filters that can be applied to
            subscriptions.";
        leaf filter-id {
            type filter-id;
            description
                "An identifier to differentiate between filters.";
        }
        uses notif:base-filter;
    }
}
container subscription-config {
    if-feature "configured-subscriptions";
    description
        "Contains the list of subscriptions that are configured,
        as opposed to established via RPC or other means.";
    list subscription {
        key "subscription-id";
        description
            "Content of a subscription.";
        leaf subscription-id {
            type subscription-id;
        }
    }
}

```

---

```
        description
            "Identifier to use for this subscription.";
    }
    uses subscription-info;
    uses receiver-info {
        if-feature "configured-subscriptions";
    }
    uses push-source-info {
        if-feature "configured-subscriptions";
    }
}
}
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, e.g. create-subscription, delete-subscription,
        and modify-subscription, as well as subscriptions that
        have been established via configuration.";
    list subscription {
        key "subscription-id";
        config false;
        description
            "Content of a subscription.
            Subscriptions can be created using a control channel
            or RPC, or be established through configuration.";
        leaf subscription-id {
            type subscription-id;
            description
                "Identifier of this subscription.";
        }

        leaf configured-subscription {
            if-feature "configured-subscriptions";
            type empty;
            description
                "The presence of this leaf indicates that the
                subscription originated from configuration, not
                through a control channel or RPC.";
        }

        leaf subscription-status {
            type subscription-status;
        }
    }
}
```

```
description
  "The status of the subscription.";
```

```
    }
    uses subscription-info;
    uses receiver-info {
      if-feature "configured-subscriptions";
    }
    uses push-source-info {
      if-feature "configured-subscriptions";
    }
  }
}
```

<CODE ENDS>

## 9. Backwards Compatibility

Capabilities are advertised in messages sent by each peer during session establishment [[RFC6241](#)]. Servers supporting the features in this document must advertise both capabilities "urn:ietf:params:netconf:capability:notification:1.0" and "urn:ietf:params:netconf:capability:notification:1.1".

An example of a hello message by a server during session establishment would be:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:netconf:base:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:startup:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:notification:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:notification:1.1
    </capability>
  </capabilities>
</hello>
```

```
</capabilities>
<session-id>4</session-id>
</hello>
```

Figure 9: Hello message

Clients that only support [[RFC5277](#)] recognize capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore capability "urn:ietf:params:netconf:capability:notification:1.1".

This allows them interacting with the server as per [[RFC5277](#)]. Clients that support the features in this document recognize both capabilities. This allows them interacting with the server as per this document.

Note that to support backwards compatibility, the yang models in this document include two types of naming conventions. That used in [[RFC5277](#)], e.g., replayComplete; and that commonly used in yang models, e.g., subscription-started.

## 10. Security Considerations

The security considerations from the base NETCONF document [[RFC6241](#)] also apply to the notification capability.

The <notification> elements are never sent before the transport layer and the NETCONF layer, including capabilities exchange, have been established and the manager has been identified and authenticated.

A secure transport must be used and the server must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of NETCONF content involved. When a <get> is received that refers to the content defined in this memo, clients should only be able to view the content for which they have sufficient privileges. <create-subscription> and <establish-subscription> operations can be considered like deferred <get>, and the content that different users can access may vary. This different access is reflected in the <notification> that different users are able to subscribe to.

The contents of notifications, as well as the names of event streams, may contain sensitive information and care should be taken to ensure

that they are viewed only by authorized users. The NETCONF server MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy NETCONF client sends a number of <create-subscription> requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the suspect underlying NETCONF sessions using the <kill-session> operation. If the client uses <establish-subscription>, the server can also suspend or terminate subscriptions with per-subscription granularity.

A subscription could be configured on another receiver's behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver, which doesn't even support subscriptions. Clients that do not want pushed data need

only terminate or refuse any transport sessions from the publisher. In addition, the NETCONF Authorization Control Model [[RFC6536](#)] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

Note that streams can define additional authorization requirements. For instance, in [[I-D.ietf-netconf-yang-push](#)], each of the elements in its data plane notifications must also go through access control.

## 11. Issues that are currently being worked and resolved

### 11.1. Unresolved and yet-to-be addressed issues

EN1 - Definition of basic set of Stream types. What streams are provided and what do they contain (includes default 5277 stream).

EN2 - Clarify interplay between filter definitions and different streams. Includes information in subtrees of event payloads.

EN3 - Mechanisms for diagnostics, e.g. deal with dropped updates, monitoring when they occur, etc

EN4 - How to allow for seamless integration with non-standard

encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

EN7 - Detecting loss of a sequential update notification, and mechanisms to resend. Implications to transports must be thought through.

### 11.2. Agreement in principal

EN6 - Stream discovery. Allow to discover additional stream properties.

EN9 - Multiple receivers per Configured Subscription is ok.

EN10 - Replay support will be provided for selected stream types (modify vs. delete)

EN11 - Required layering security requirements/considerations will be added into the YANG model for Configured Subscriptions. It will be up to the transport to meet these requirements.

EN12 - Test-only option for a subscription is desired. But it still needs to be defined.

EN13 - [RFC6241](#) Subtree-filter definition in 5277bis cannot apply to elements of an event. Must explicitly define how 6241 doesn't apply filtering within a 5277bis event.

EN14 - Ensure that Configured Subscriptions are fully defined in YANG model.

### 11.3. Resolved Issues

EN5 - This draft obsoletes 5277, as opposed to being in parallel with it

EN8 - No mandatory transport

EN15 - Term for Dynamic and Static Subscriptions (move to "Configured")

## 12. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Yang Geng, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, 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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", [RFC 5277](#), DOI 10.17487/RFC5277, July 2008, <<http://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, <<http://www.rfc-editor.org/info/rfc6020>>.

Clemm, et al.

Expires March 15, 2017

[Page 42]

---

Internet-Draft

Event Notifications

September 2016

- [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, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", [RFC 6536](#), DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

### 13.2. Informative References

[I-D.ietf-netconf-netconf-event-notif]

Gonzalez Prieto, Alberto., Clemm, Alexander., Voit, Eric., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", August 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>>.

[I-D.ietf-netconf-restconf]

Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D [draft-ietf-netconf-restconf-13](#), April 2016.

[I-D.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., Gonzalez Prieto, Alberto., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", June 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

#### Authors' Addresses

Alexander Clemm  
Cisco Systems

Email: ludwig@clemm.org

Alberto Gonzalez Prieto  
Cisco Systems

Email: albertgo@cisco.com

Eric Voit  
Cisco Systems

Email: [evoit@cisco.com](mailto:evoit@cisco.com)

Einar Nilsen-Nygaard  
Cisco Systems

Email: [einarnn@cisco.com](mailto:einarnn@cisco.com)

Ambika Prasad Tripathy  
Cisco Systems

Email: [ambtripa@cisco.com](mailto:ambtripa@cisco.com)

Sharon Chisholm  
Ciena

Email: [schishol@ciena.com](mailto:schishol@ciena.com)

Hector Trevino  
Cisco Systems

Email: [htrevino@cisco.com](mailto:htrevino@cisco.com)