Network Working Group
Internet-Draft
Expires: October 30, 2006

S. Chisholm
K. Curran
Nortel
H. Trevino
Cisco

April 28, 2006

NETCONF Event Notifications
draft-ietf-netconf-notification-01.txt

Status of this Memo

Copyright Notice

Abstract

This memo defines a framework for sending asynchronous messages, or event notifications in NETCONF. It defines both the operations necessary to support this concept, and also discusses implications for the mapping to application protocols.

Table of Contents

1. Introduction

NETCONF [NETCONF-PROTO] can be conceptually partitioned into four layers:

```
   Layer                      Example
    +-------------+      +----------------------------------------+
    |   Content   |      |     Configuration data                 |
    +-------------+      +----------------------------------------+
              |                            |
    +-------------+      +-------------------------------------------+
    | Operations  |      | <get-config>, <edit-config> <notification>|
    +-------------+      +-------------------------------------------+
              |                            |                    |
    +-------------+      +-----------------------------+       |
    |     RPC     |      |    <rpc>, <rpc-reply>       |       |
    +-------------+      +-----------------------------+       |
             |                           |                     |
    +-------------+      +------------------------------------------+
    | Application |      |   BEEP, SSH, SSL, console                |
    |   Protocol  |      |                                          |
    +-------------+      +------------------------------------------+
```

This document defines a framework for sending asynchronous messages, or event notifications in NETCONF. It defines both the operations necessary to support this concept, and also discusses implications for the mapping to application protocols.

Figure 1

1.1 Definition of Terms

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 [3].

Element: An XML Element[XML].

Managed Entity: A node, which supports NETCONF[NETCONF] and has access to management instrumentation. This is also known as the NETCONF server.

Managed Object: A collection of one of more Elements that define an abstract thing of interest.

1.2 Event Notifications in NETCONF

An event is something that happens which may be of interest - a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system, for example. Often this results in an asynchronous message, sometimes referred to as a notification or event notification, being sent out to interested parties to notify them that this event has occurred.

This memo defines a mechanism whereby the NETCONF client indicates interest in receiving event notifications from a NETCONF server by creating a subscription to receive event notifications. The NETCONF server replies to indicate whether the subscription request was successful and, if it was successful, begins sending the event notifications to the NETCONF client as the events occur within the system. These event notifications will continue to be sent until either the NETCONF session is terminated or an explicit command to cancel the subscription is sent. The event notification subscription allows a number of options to enable the NETCONF client to specify which events are of interest. These are specified when the subscription is created, but can be modified later using a modify subscription command.

2. Event-Related Operations

2.1 Subscribing to receive Events

The event notification subscription is initiated by the NETCONF client and responded to by the NETCONF server. When the event notification subscription is created, the events of interest are specified.

It is possible to create more than one event notification subscription on a single underlying connection. Each event notification subscription therefore has its own unique identifier.

Content for an event notification subscription can be selected by specifying which event classes are of interest and /or by applying user-specified filters.

2.1.1 create-subscription

<create-subscription>

Description:

This command initiates an event notification subscription which will send asynchronous event notifications to the initiator of the command until the <cancel-subscription > command is sent.

Parameters:

Event Classes:

An optional parameter that indicates which event classes are of interest. If not present, events of all classes will be sent.

Filter:

An optional parameter that indicates which subset of all possible events are of interest. The format of this parameter is the same as that of the filter parameter in the NETCONF protocol operations. If not present, all events not precluded by other parameters will be sent. These filter parameters can only be modified using the modify-subscription command.

Named Profile

An optional parameter that points to a separately defined filter profile. The contents of the profile are specified in the provided XML Schema. If not present, no additional filtering will be applied. If the separate definition of these filters is updated, then these changes will be reflected in the filtered events on this subscription.

Positive Response:

If the NETCONF server can satisfy the request, the server sends an <rpc-reply> element containing a <data> element containing the subscription ID.

Negative Response:

An <rpc-error> element is included within the <rpc-reply> if the request cannot be completed for any reason.

2.2 Sending Event Notifications

Once the subscription has been set up, the NETCONF server sends the event notifications asynchronously along the connection. Notifications are tagged with event classes, subscription ID, sequence number, and date and time.

2.2.1 Event Notification

<notification>

Description:

An event notification is sent to the initiator of an <create-subscription> command asynchronously when an event of interest (i.e. meeting the specified filtering criteria) to them has occurred. An event notification is a complete XML document.

Parameters:

Event Classes:

The event class or classes associated with this event notification

Subscription Id:

A unique identifier for this event subscription

Sequence Number:

A sequentially increasing number to uniquely identify event notifications for this subscription. It starts at 0, always increases by just one and rolls back to 0 after its maximum value is reached.

Date and Time:

The date and time that the event notification was sent by the NETCONF server.

Positive Response:

No response.

Negative Response:

No response.

2.2.1.1 **Event Notification**

The NETCONF Event notification structure is shown in the following figure.

```
   ___________________________________________________________________
   || Notification Header                                      || Data |
   ||__________________________________________________________||______|
   || subscriptionId| eventClasses| sequenceNumber| dateAndTime||      |
   ||_______________|_____________|_______________|____________||______|
```

2.3 **Changing the Subscription**

After an event notification subscription has been established, the NETCONF client can initiate a request to change properties of the event notification subscription. This prevents loss of event notifications that might otherwise occur during a cancelling and recreation of the event notification subscription. This command is responded to by the NETCONF server

2.3.1 modify-subscription

<modify-subscription>

Description:

Change properties of the event notification subscription.

Parameters:

Subscription Id:

A unique identifier for this event subscription.

Event Classes:

An optional parameter that indicates which Event Classes are of interest. If not present, events of all classes will be sent.

Filter:

An optional parameter that indicates which subset of all possible events that are of interest. The format is the same filter used for other NETCONF commands. If not present, all events not precluded by other parameters will be sent. These filter parameters can only be modified using the modify-subscription command.

Named Profile:

An optional parameter that points to separately defined filter profile. The contents of the profile are specified in provided XML Schema. If not present, no additional filtering will be applied. If the separate definition of these filters is updated, then these changes will be reflected in the events seen on this subscription.

Positive Response:

If the NETCONF server was able to satisfy the request, an <rpc-reply> is sent that includes an <ok> element.

Negative Response:

An <rpc-error> element is included within the <rpc-reply> if the request cannot be completed for any reason.

2.4 Terminating the Subscription

Closing of the event notification subscription is initiated by the NETCONF client. The specific subscription to be closed is specified using a subscription ID. The NETCONF server responds. Note that the NETCONF session may also be torn down for other reasons and this will also result in the subscription being cancelled, but is not subjected to the behaviour of this command.

2.4.1 cancel-subscription

<cancel-subscription>

Description:

Stop and delete the event notification subscription.

Parameters:

Subscription Id:

A unique identifier for this event notification subscription.

Positive Response:

If the NETCONF server was able to satisfy the request, an <rpc-reply> is sent that includes an <ok> element.

Negative Response:

An <rpc-error> element is included within the <rpc-reply> if the request cannot be completed for any reason.

3. Supporting Concepts

3.1 Capabilities Exchange

The ability to process and send event notifications is advertised during the capability exchange between the NETCONF client and server.

"urn:ietf:params:xml:ns:netconf:notification:1.0"

For Example

```
      <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:xml:ns:netconf:capability:startup:1.0
          </capability>
          <capability>
            urn:ietf:params:xml:ns:netconf:notification:1.0
          </capability>
        </capabilities>
        <session-id>4</session-id>
      </hello>
```

3.2 Querying Subscription Properties

The following Schema can be used to retrieve information about active event notification subscriptions

```
    <xs:schema
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:nsub="urn:ietf:params:xml:ns:netconf:subscription:1.0"
      targetNamespace= "urn:ietf:params:xml:ns:netconf:subscription:1.0"
      xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0"
      xmlns:ncEvent= "urn:ietf:params:xml:ns:netconf:notification:1.0"
     xmlns:nm="urn:ietf:params:xml:ns:netconf:appInfo:1.0"
     elementFormDefault="qualified" attributeFormDefault="unqualified"
             xml:lang="en">
        <xs:annotation>
                       <xs:documentation xml:lang="en">
                     Schema for reporting on Event Subscriptions
         </xs:documentation>
```

```
         <xs:appinfo>
           <nm:identity
               xmlns:nm="urn:ietf:params:xml:ns:netmod:base:1.0">
               <nm:Name>NetConfStateSchema</nm:Name>
               <nm:LastUpdated>2006-04-30T09:30:47-05:00
               </nm:LastUpdated>
               <nm:Organization>IETF</nm:Organization>
               <nm:Description>
                  A schema that can be used to learn about current
                  NetConf Event subscriptions and creating named
                  profiles
               </nm:Description>
             </nm:identity>
           </xs:appinfo>
              </xs:annotation>

       <xs:import namespace="http://www.w3.org/XML/1998/namespace"
                  schemaLocation="http://www.w3.org/2001/xml.xsd"/>
       <xs:import
            namespace="urn:ietf:params:xml:ns:netconf:notifications:1.0"
            schemaLocation="draft-ietf-netconf-notification-01.xsd"/>
       <xs:import namespace="urn:ietf:params:xml:ns:netconf:base:1.0"
                   schemaLocation="draft-ietf-netconf-prot-12.xsd"/>


      <xs:element name="netconfSubscription">
        <xs:annotation>
            <xs:appinfo>
              <nm:minAccess><read/></nm:minAccess>
              <nm:maxAccess><read/></nm:maxAccess>
            </xs:appinfo>
        </xs:annotation>
        <xs:complexType>
        <xs:sequence maxOccurs="unbounded">

          <xs:element name="session-id"
                           type="netconf:SessionId" >
            <xs:annotation>
              <xs:documentation xml:lang="en">
              The session id associated with this subscription.
              </xs:documentation>
           </xs:annotation>
          </xs:element>

                 <xs:element name="subscriptionID"
                           type="ncEvent:SubscriptionID" >
            <xs:annotation>
              <xs:documentation xml:lang="en">
```

```
                The subscription id associated with this subscription.
                </xs:documentation>
             </xs:annotation>
            </xs:element>

            <xs:element name="eventClasses">
              <xs:annotation>
                <xs:documentation xml:lang="en">
                The event classes associated with this subscription.
                </xs:documentation>
             </xs:annotation>
             <xs:complexType>
               <xs:sequence minOccurs="0" maxOccurs="unbounded">
                 <xs:element ref="ncEvent:EventClass"/>
               </xs:sequence>
             </xs:complexType>
            </xs:element>

              <xs:element name="filter"
                         type="netconf:filterInlineType"  minOccurs="0">
              <xs:annotation>
                <xs:documentation xml:lang="en">
                The filters associated with this subscription.
                </xs:documentation>
             </xs:annotation>
            </xs:element>

            <xs:element name="namedProfile"
                  type="xs:string" minOccurs="0">
                           <xs:annotation>
                <xs:documentation xml:lang="en">
                The named profile associated with this subscription. Note
                 that the  contents of the named profile may have changed
               since it was last applied.
                </xs:documentation>
                           </xs:annotation>
                <xs:keyref name="namedProfileKeyRef"
                           refer="nsub:namedProfileKey">
                  <xs:selector xpath=".//namedProfile"/>
                  <xs:field xpath="namedProfile"/>
                </xs:keyref>
            </xs:element>

            <xs:element name="lastModified"
                  type="xs:dateTime" >
                           <xs:annotation>
                <xs:documentation xml:lang="en">
                The last time this subscription was modified. If it has
```

```
              not been modified since creation, this is the time of
             subscription creation.
              </xs:documentation>
           </xs:annotation>
           </xs:element>

           <xs:element name="messagesSent"
                type="xs:integer" minOccurs="0">
                         <xs:annotation>
              <xs:documentation xml:lang="en">
              A count of event notifications sent along this connection
              since the subscription was created.
              </xs:documentation>
           </xs:annotation>
           </xs:element>

           <xs:element name="lastSequenceNumber"
                type="xs:integer" minOccurs="0">
                         <xs:annotation>
              <xs:documentation xml:lang="en">
              The sequence number of the last event notification sent to
               this subscription
              </xs:documentation>
           </xs:annotation>
           </xs:element>
          <xs:element name="key">
            <xs:key name="uniqueSubscription">
              <xs:selector xpath=".//subscription"/>
              <xs:field xpath="session-id"/>
              <xs:field xpath="subscriptionID"/>
            </xs:key>
          </xs:element>
         </xs:sequence>
         </xs:complexType>
         </xs:element>

     <xs:element name="netconfSubscriptions">
       <xs:complexType>
       <xs:sequence>
         <xs:element ref="nsub:netconfSubscription" minOccurs="0"
                                 maxOccurs="unbounded" />
         </xs:sequence>
       </xs:complexType>
     </xs:element>


      <xs:element name="namedProfile">
        <xs:annotation>
```

```
          <xs:appinfo>
            <nm:minAccess><read/></nm:minAccess>
            <nm:maxAccess><read/> <write/> <create/> <delete/>
             </nm:maxAccess>
          </xs:appinfo>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
              <xs:element name="name"/>
              <xs:element name="eventClasses">
                <xs:annotation>
                  <xs:documentation xml:lang="en">
                    The event classes associated with this named
                     Profile.
                  </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence minOccurs="0" maxOccurs="unbounded">
                    <xs:element ref="ncEvent:EventClass"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>

              <xs:element name="filter"
                type="netconf:filterInlineType"  minOccurs="0">
                <xs:annotation>
                  <xs:documentation xml:lang="en">
                    The filters associated with this named Profile.
                  </xs:documentation>
                </xs:annotation>
              </xs:element>

              <xs:element name="lastModified" type="xs:dateTime">
                <xs:annotation>
                  <xs:documentation>
                    The timestamp of the last modification to this
                     named Profile. Note that modification of the
                     profile does not cause an immediate update
                     to all applicable subscription. Therefore, this
                     time should be compared with the last
                     modified time associated with the subscription.
                     If this time is earlier, then the subscription
                      is using the exact set of parameters associated
                      with this named profile.  If this time is
                     later, then the subscription is using an earlier
                     version of this named profile and the exact
                     parameters may not match.
                  </xs:documentation>
```

```
                  <xs:appinfo>
                    <nm:minAccess><read/></nm:minAccess>
                    <nm:maxAccess><read/> </nm:maxAccess>
                  </xs:appinfo>
                </xs:annotation>
              </xs:element>

              <xs:element name="key">
                  <xs:key name="namedProfileKey">
                    <xs:selector xpath="*/name" />
                    <xs:field xpath="name" />
                  </xs:key>
              </xs:element>
            </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="namedProfiles">
         <xs:complexType>
          <xs:sequence>
            <xs:element ref="nsub:namedProfile" minOccurs="0"
                                  maxOccurs="unbounded" />
          </xs:sequence>
           </xs:complexType>
       </xs:element>
       </xs:schema>
```

3.3 One-way Notification Messages

In order to support the concept that each individual event notification is a well-defined XML-document that can be processed without waiting for all events to come in, it makes sense to define events, not as an endless reply to a subscription command, but as independent messages that originate from the NETCONF server. In order to support this model, this memo introduces the concept of notifications, which are one-way messages.

A one-way message is similar to the two-way RPC message, except that no response is expected to the command. In the case of event notification, this message will originate from the NETCONF server, and not the NETCONF client.

3.4 Filter Dependencies

Note that when multiple filters are specified (Event Class, in-line

Filter, Named Profiles), they are applied collectively, so event notifications needs to pass all specified filters in order to be sent to the subscriber. If a filter is specified to look for data of a particular value, and the data item is not present within a particular event notification for its value to be checked against, it will be filtered out. For example, if one were to check for 'severity=critical' in a configuration event notification where this field was not supported, then the notification would be filtered out.

3.4.1 Named Profiles

A named profile is a filter that is created ahead of time and applied at the time an event notification subscription is created or modified. Note that changes to the profile after the subscription has been created will have no effect unless a modify subscription command is issued. Since named profiles exist outside of the subscription, they persist after the subscription has been cancelled.

3.4.2 Filtering

Just-in-time filtering is explicitly stated when the event notification subscription is created. These filters can only be changed using the modify subscription command. This is specified via the Filter parameter. Filters only exist as parameters to the subscription.

3.5 Event Classes

Events can be broadly classified into one more event classes. Each event class identifies a set of event notifications which share important characteristics, such being generated from similar events or sharing much of the same content.

The initial set of event classes is fault, configuration, state, audit, data, maintenance, metrics, security, information, heartbeat and syslog.

A fault event notification is generated when a fault condition (error or warning) occurs. A fault event may result in an alarm. Examples of fault events could be a communications alarm, environmental alarm, equipment alarm, processing error alarm, quality of service alarm, or a threshold crossing event. See RFC3877 and RFC2819 for more information.

A configuration event, alternatively known as an inventory event, is used to notify that hardware, software, or a service has been added/ changed/removed. In keeping aligned with NETCONF protocol operations, configuration events may included copy configuration

event, delete configuration event, or the edit configuration event (create, delete, merge, replace).

A state event indicates a change from one state to another, where a state is a condition or stage in the existence of a managed entity. State change events are seen in many specifications. For Entity state changes, see [Entity-State-MIB] for more information.

Audit events provide event of very specific actions within a managed device. In isolation an audit events provides very limited data. A collection of audit information forms an audit trail.

A data dump event is an asynchronous event containing information about a system, its configuration, state, etc.

A maintenance event signals the beginning, process or end of an action either generated by a manual or automated maintenance action.

A metrics event contains a metric or a collection of metrics. This includes performance metrics.

A heart beat event is sent periodically to enable testing that the communications channel is still functional. It behaves much like the other event classes, with the exception that implementations may not want to include an event log, if supported. Although widely used throughout the industry, no current corresponding work within the IETF. However, other standards bodies such as the TeleManagement Forum have similar definitions.

An Information event is something that happens of interest which is within the expected operational behaviour and not otherwise covered by another class.

The syslog event class is used to indicate tunneled syslog content. The content and format of the message will be compliant to syslog standards.

3.6 Defining Event Notifications

Event Notifications are defined ahead of time by defining an XML element and assigning it to particular event classes. This will be done using an "eventClasses" attribute.

3.7 Interleaving Messages

While each NETCONF message must be a complete XML document, the design of the event system allows for the interleaving of complete asynchronous event notifications with complete synchronous messages.

It is possible to still send command-response type messages such as <modify-subscription> while events are being generated. The only restriction is that each message must be complete

The following sequence diagram demonstrates an example NETCONF session where after basic session establishment and capability exchange, NETCONF client (C), subscribes to receive event notifications. The NETCONF server (S), starts sending event notifications as events of interest happen within the system. The NETCONF client decides to change the characteristics of their event subscription by sending a <modify-subscription> command. Before the NETCONF server, receives this command, another event is generated and the NETCONF server starts to send the event notification. The NETCONF server finishes sending this event notification before processing the <modify-subscription> command and sending the reply.

```
                          C                            S
                          |                            |
                          |  capability exchange       |
                          |--------------------------->|
                          |<-------------------------->|
                          |                            |
                          |  <create-subscription>     |
                          |--------------------------->|
                          |<---------------------------|
                          |                            |
                          |     <notification>         |
                          |<---------------------------|
                          |                            |
                          |     <notification>         |
                          |<---------------------------|
                          |                            |
                          |  <modify-subscription>     |
                          |--------------------------->| (buffered)
                          |     <notification>         |
                          |<---------------------------|
                          |  <rpc-reply>               |
                          |<---------------------------|
```

4. XML Schema for Event Notifications

```
   <?xml version="1.0" encoding="UTF-8"?>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"
             xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0"
       targetNamespace="urn:ietf:params:xml:ns:netconf:notification:1.0"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified"
               xml:lang="en">
        <!--
          import standard XML definitions
          -->
        <xs:import namespace="http://www.w3.org/XML/1998/namespace"
                   schemaLocation="http://www.w3.org/2001/xml.xsd">
          <xs:annotation>
            <xs:documentation>
              This import accesses the xml: attribute groups for the
              xml:lang as declared on the error-message element.
            </xs:documentation>
          </xs:annotation>
        </xs:import>

        <!-- import base netconf definitions -->
    <xs:import namespace="urn:ietf:params:xml:ns:netconf:base:1.0"
          schemaLocation="urn:ietf:params:xml:ns:netconf:base:1.0" />


   <!-- ************** Type definitions ***********************-->

        <xs:simpleType name="SubscriptionID">
        <xs:annotation>
          <xs:documentation>
          The unique identifier for this particular subscription within
          the session.
          </xs:documentation>
          </xs:annotation>
           <xs:restriction base="xs:string"/>
           </xs:simpleType>

           <xs:simpleType name="SequenceNumber">
        <xs:annotation>
          <xs:documentation>
          A monotonically  increasing integer. Starts at 0.
          Always increases by just one. Roll back to 0 after maximum
          value is reached.
          </xs:documentation>
```

```
          </xs:annotation>
           <xs:restriction base="xs:integer"/>
           </xs:simpleType>

           <xs:complexType name="EventClassType"/>
           <xs:element name="EventClass"
                        type="EventClassType" abstract="true"/>
           <xs:element name="fault" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="information" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="state" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="configuration" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="data" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="maintenance" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="metrics" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="security" type="EventClassType"
                        substitutionGroup="EventClass"/>
           <xs:element name="heartbeat" type="EventClassType"
                        substitutionGroup="EventClass"/>

         <xs:complexType name="EventClasses">
           <xs:sequence maxOccurs="unbounded">
             <xs:element ref="EventClasses" />
           </xs:sequence>
         </xs:complexType>



   <!-- ************** Symmetrical Operations  ********************-->


        <!--
          <create-subscription> operation
          -->
        <xs:complexType name="createSubscriptionType">
          <xs:complexContent>
            <xs:extension base="netconf:rpcOperationType">
              <xs:sequence>
                <xs:element name="event-classes"
                                      minOccurs="0">
                  <xs:complexType>
                     <xs:complexContent>
```

```
                  <xs:extension base="EventClasses"/>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
           <xs:element name="filter"
                type="netconf:filterInlineType" minOccurs="0"/>
           <xs:element name="named-profile"
                       type="xs:string" minOccurs="0"/>
         </xs:sequence>
       </xs:extension>
     </xs:complexContent>
   </xs:complexType>
   <xs:element name="create-subscription"
               type="createSubscriptionType"
               substitutionGroup="netconf:rpcOperation"/>

   <!--
     <modify-subscription> operation
     -->
   <xs:complexType name="modifySubscriptionType">
     <xs:complexContent>
       <xs:extension base="netconf:rpcOperationType">
         <xs:sequence>
           <xs:element name="subscription-id"
                            type="SubscriptionID" />
           <xs:element name="event-classes"
                                 minOccurs="0">
             <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="EventClasses"/>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
           <xs:element name="filter"
                       type="netconf:filterInlineType"
                       minOccurs="0"/>
           <xs:element name="named-profile"
                       type="xs:string" minOccurs="0"/>
         </xs:sequence>
       </xs:extension>
     </xs:complexContent>
   </xs:complexType>
   <xs:element name="modify-subscription"
               type="modifySubscriptionType"
               substitutionGroup="netconf:rpcOperation"/>

   <!--
     <cancel-subscription> operation
```

```
          -->
        <xs:complexType name="cancelSubscriptionType">
          <xs:complexContent>
            <xs:extension base="netconf:rpcOperationType">
              <xs:sequence>
                <xs:element name="subscription-id"
                     type="SubscriptionID" />
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:element name="cancel-subscription"
                    type="cancelSubscriptionType"
                    substitutionGroup="netconf:rpcOperation"/>


   <!-- ************** One-way Operations  ******************-->

          <!--
          <Event> operation
          -->
        <xs:complexType name="NotificationType">
          <xs:complexContent>
              <xs:sequence>
                <xs:element name="subscription-id"
                                     type="SubscriptionID"/>
                <xs:element name="event-classes" type="EventClasses"/>
                <xs:element name="sequence-number"
                                  type="SequenceNumber"/>
                <xs:element name="date-time" type="xs:dateTime">
                   <xs:annotation>
                      <xs:documentation>
                      The date and time that the event notification was
                      sent by the netconf server.
                      </xs:documentation>
                   </xs:annotation>
                </xs:element>
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
        <xs:element name="notification" type="NotificationType"/>

      </xs:schema>
```

5. Mapping to Application Protocols

Currently, the NETCONF family of specification allows for running NETCONF over a number of application protocols, some of which support multiple configurations. Some of these options will be better suited for supporting event notifications then others.

5.1 SSH

Session establishment and two-way messages are based on the NETCONF over SSH transport mapping [NETCONF-SSH]

One-way event messages are supported as follows: Once the session has been established and capabilities have been exchanged, the server may send complete XML documents to the NETCONF client containing notification elements. No response is expected from the NETCONF client.

As the other examples in [NETCONF-SSH] illustrate, a special character sequence, MUST be sent by both the client and the server after each XML document in the NETCONF exchange. This character sequence cannot legally appear in an XML document, so it can be unambiguously used to identify the end of the current document in the event notification of an XML syntax or parsing error, allowing resynchronization of the NETCONF exchange.

The NETCONF over SSH session to receive an event notification might look like the following. Note the event notification contents (delimited by <data> </data> tags) are not defined in this document and are provided herein simply for illustration purposes:

```
       <?xml version="1.0" encoding="UTF-8"?>
          <notification
                xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
            <subscription-id>123456</subscription-id>
            <event-class><configuration/><audit/></event-classes>
            <sequence-number>2</sequence-number>
            <date-time>2000-01-12T12:13:14Z</date-time>
              <data>
                 <user>Fred Flinstone</user>
                 <operation>
                  <edit-config>
                    <target>
                     <running/>
                    </target>
                    <config>
                      <top xmlns="http://example.com/schema/1.2/config">
                         <interface>
                           <name>Ethernet0/0</name>
                           <mtu>1500</mtu>
                        </interface>
                      </top>
                   </config>
                 </edit-config>
               </operation>
             </data>
          </notification>
        ]]>
    ]]>
```

5.2 BEEP

Session establishment and two-way messages are based on the NETCONF over BEEP transport mapping NETCONF-BEEP

5.2.1 One-way Notification Messages in Beep

One-way notification messages can be supported either by mapping to the existing one-to-many BEEP construct or by creating a new one-to-none construct.

This area is for future study.

5.2.1.1 One-way messages via the One-to-many Construct

Messages in one-to-many exchanges: "rpc", "notification", "rpc-reply"

Messages in positive replies: "rpc-reply", "rpc-one-way"

5.2.1.2 One-way notification messages via the One-to-none Construct

Note that this construct would need to be added to an extension or update to 'The Blocks Extensible Exchange Protocol Core' RFC 3080.

MSG/NoANS: the client sends a "MSG" message, the server, sends no reply.

In one-to-none exchanges, no reply to the "MSG" message is expected.

5.3 SOAP

Session management and message exchange are based on the NETCONF over SOAP transport mapping NETCONF-SOAP

Note that the use of "persistent connections" "chunked transfer-coding" when using HTTP becomes even more important in the supporting of event notifications

5.3.1 A NETCONF over Soap over HTTP Example

```
C: POST /netconf HTTP/1.1
C: Host: netconfdevice
C: Content-Type: text/xml; charset=utf-8
C: Accept: application/soap+xml, text/*
C: Cache-Control: no-cache
C: Pragma: no-cache
C: Content-Length: 465
C:
C: <?xml version="1.0" encoding="UTF-8"?>
C: <soapenv:Envelope
C:   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
C:   <soapenv:Body>
C:     <rpc message-id="101"
C:        xmlns=
        "xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
C:       <create-subscription>
C:       </create-subscription>
C:     </rpc>
C:   </soapenv:Body>
C: </soapenv:Envelope>
```

The response:

```
S: HTTP/1.1 200 OK
S: Content-Type: application/soap+xml; charset=utf-8
S: Content-Length: 917
S:
```

```
S: <?xml version="1.0" encoding="UTF-8"?>
S: <soapenv:Envelope
S:   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
S:   <soapenv:Body>
S:     <rpc-reply message-id="101"
S:        xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
S:       <data>
S:         <top xmlns=
              "http://example.com/schema/1.2/notification">
S:           <subscriptionId>123456</subscriptionId>
S:         </top>
S:       </data>
S:     </rpc-reply>
S:   </soapenv:Body>
S: </soapenv:Envelope>
```

And then some time later

```
S: HTTP/1.1 200 OK
S: Content-Type: application/soap+xml; charset=utf-8
S: Content-Length: 917
S:
S: <?xml version="1.0" encoding="UTF-8"?>
S: <soapenv:Envelope
S:   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
S:   <soapenv:Body>
S:     <notification
          xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
S:      <subscriptionID>123456</subscriptionID>
S:      <eventClass><configuration/><audit/></eventClass>
S:      <sequenceNumber>2</sequenceNumber>
S:           <dateAndTime>2000-01-12T12:13:14Z</dateAndTime>
S:        <data>
S:           <user>Fred Flinstone</user>
S:              <operation>
S:               <edit-config>
S:              <target>
S:               <running/>
S:              </target>
S:             <config>
S:              <top xmlns="http://example.com/schema/1.2/config">
S:                   <interface>
S:                     <name>Ethernet0/0</name>
S:                     <mtu>1500</mtu>
S:                  </interface>
S:               </top>
S:            </config>
S:           </edit-config>
```

```
      S:          </operation>
      S:        </data>
      S:     </notification>
      S:    </soapenv:Body>
      S: </soapenv:Envelope>
```

6. Filtering examples

The following section provides examples to illustrate the various methods of filtering content on an event notification subscription.

6.1 Event Classes

The following example illustrates selecting all event notifications for EventClasses fault, state or config

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
  <create-subscription>
    <eventClasses>
       <fault/>
       <state/>
       <config/>
    </eventClasses>
  </create-subscription>
</rpc>
```

6.2 Subtree Filtering

XML subtree filtering is not well suited for creating elaborate filter definitions given that it only supports equality comparisons (e.g. in the event subtree give me all event notifications which have severity=critical or severity=major or severity=minor). Nevertheless, it may be used for defining simple notification forwarding filters as shown below.

The following example illustrates selecting fault EventClass which have severities of critical, major, or minor. The filtering criteria evaluation is as follows:

((fault) & ((severity=critical) | (severity=major) | (severity = minor)))

```
        <rpc message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
          <create-subscription>
            <eventClasses>
               <fault/>
            </eventClasses>
            <netconf:filter type="subtree">
              <neb xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
                <event>
                    <severity>critical</severity>
                </event>
                <event>
                    <severity>major</severity>
                </event>
                <event>
                    <severity>minor</severity>
                </event>
              </neb>
            </netconf:filter>
          </create-subscription>
        </rpc>
```

The following example illustrates selecting fault, state, config EventClasses which have severities of critical, major, or minor and come from card Ethernet0. The filtering criteria evaluation is as follows:

```
((fault | state | config) & ((fault & severity=critical) | (fault &
severity=major) | (fault & severity = minor) | (card=Ethernet0)))
```

```
        <rpc message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
          <create-subscription>
            <eventClasses>
               <fault/>
               <state/>
               <config/>
            </eventClasses>
            <netconf:filter type="subtree">
              <neb xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
                <event>
                    <eventClasses>fault</eventClasses>
                    <severity>critical</severity>
                </event>
                <event>
                    <eventClasses>fault</eventClasses>
                    <severity>major</severity>
                </event>
                <event>
                    <eventClasses>fault</eventClasses>
                    <severity>minor</severity>
                </event>
                <event>
                    <card>Ethernet0</card>
                </event>
              </neb>
            </netconf:filter>
          </create-subscription>
        </rpc>
```

6.3 XPATH filters

The following example illustrates selecting fault EventClass which have severities of critical, major, or minor. The filtering criteria evaluation is as follows:

```
((fault) & ((severity=critical) | (severity=major) | (severity =
minor)))
```

```
        <rpc message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
          <create-subscription>
            <eventClasses>
               <fault/>
            </eventClasses>
            <netconf:filter type="xpath">
              (/event[eventClasses/fault] and
              (/event[severity="critical"] or
               /event[severity="major"] or /event[severity="minor"]))
            </netconf:filter>
          </create-subscription>
        </rpc>
```

The following example illustrates selecting fault, state, config EventClasses which have severities of critical, major, or minor and come from card Ethernet0. The filtering criteria evaluation is as follows:

```
((fault | state | config) & ((fault & severity=critical) | (fault &
severity=major) | (fault & severity = minor) | (card=Ethernet0)))
```

```
        <rpc message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:event:1.0">
          <create-subscription>
            <eventClasses>
               <fault/>
               <state/>
               <config/>
            </eventClasses>
            <netconf:filter type="xpath">
               ((/event[eventClasses/fault]  or
               /event[eventClasses/state]     or
                /event[eventClasses/config]) and
                ( (/event[eventClasses/fault] and
                /event[severity="critical"]) or
                (/event[eventClasses/fault]    and
                /event[severity="major"])    or
                (/event[eventClasses/fault]    and
                /event[severity="minor"])    or
                /event[card="Ethernet0"]))
            </netconf:filter>
          </create-subscription>
        </rpc>
```

7. Additional Capabilities

7.1 Call-Home Notifications

7.1.1 Overview

Call-Home Notifications are an alternative model for providing notifications that may be preferred for two particular use cases. The first use case is NAT traversal as in this model, the Netconf server initiates the Notification session. The second use case is when a manager has a large number of low-priority devices that it only wants to deal with when there a known issue. While this risks loss of information, for this particular use case, this is not considered an issue. The Call-home-Notification feature supports the concept of a short-lived notification session that only exists when there is something to report.

In this feature, a subscription consists of a named profile, and an association with a Netconf client. Unlike normal subscriptions, which only exist when they are active, these subscriptions live while both dormant and active. When an event of interest happens on the managed resource, the Netconf server checks the list of dormant subscriptions and if the filtering parameters in the subscription indicate interest in the Notification resulting from the event, then the Netconf server initiates the connection to the specific Netconf client and sends the Notification. When the Notification has been sent, the connection is terminated.

7.1.1.1 Session Lifecycle

In order to avoid situations in which a sessions is continuously setup and torn down, an inactivity timer is configured on the server. The timeout interval value is the same for all sessions (i.e. system wide) and each session has its own timer. Upon expiration of the inactivity timer, the connection is terminated, otherwise if activity is detected, the timer is reset.

[Editor's note: alternatives here were to either create and tear down the session for each notification received or to have the server somehow figure out that there are more notifications coming soon after it has sent a notification and therefore keeps the connection up.]

The session establishment procedure is as follows:

1) The NETCONF server initiates a session using a recognized application protocol (SSH, Beep, SOAP, etc). In order to "activate" this reverse behaviour a new SSH subsystem may need to be defined.

This is for further study. In addition, the NE hosting the NETCONF server must support both client and server modes in the case of SSH.

2)   Client and server are authenticated according to the underlying application protocol (e.g. SSH, BEEP)

3)   If using BEEP, as described in [NETCONF-BEEP] either party may initiate the BEEP session. Once this occurs, the assumption is that both parties know their roles. At this point, the NETCONF client, initiates NETCONF session establishment whether running SSH or BEEP.

7.1.2 Dependencies

This feature is dependant on the named profiles concept from the normal subscription method as well as the definition of <notification>.

It also uses the same <notification>

7.1.3 Capability Identifier

urn:ietf:params:xml:ns:netconf:callHomeNotification:1.0

7.1.3.1 New Operations

7.1.3.1.1 New Data Model

```
   <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:nsub="urn:ietf:params:xml:ns:netconf:subscription:1.0"
      targetNamespace=
              "urn:ietf:params:xml:ns:netconf:callHomeSubscription:1.0"
      xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0"
      xmlns:ncEvent= "urn:ietf:params:xml:ns:netconf:event:1.0"
      xmlns:nm="urn:ietf:params:xml:ns:netconf:appInfo:1.0"
      elementFormDefault="qualified"
                     attributeFormDefault="unqualified" xml:lang="en">
   <xs:annotation>
     <xs:documentation xml:lang="en">
          Schema for reporting on dormant Call-Home Notification
          Subscriptions
     </xs:documentation>
      <xs:appinfo>
          <nm:identity
                  xmlns:nm="urn:ietf:params:xml:ns:netmod:base:1.0">
                <nm:Name>NetConfCallHomeSchema</nm:Name>
                 <nm:LastUpdated>2006-04-30T09:30:47-05:00
```

```
                 </nm:LastUpdated>
                 <nm:Organization>IETF</nm:Organization>
                  <nm:Description>
                       A schema that can be used to learn about callHome
                       Notification subscriptions
                   </nm:Description>
            </nm:identity>
     </xs:appinfo>
    </xs:annotation>

   <xs:import
      namespace="urn:ietf:params:xml:ns:netconf:subscription:1.0"
      schemaLocation="urn:ietf:params:xml:ns:netconf:subscription:1.0"/>


    <xs:element name="callHomeSubscription">
        <xs:annotation>
               <xs:appinfo>
                    <nm:minAccess><read/></nm:minAccess>
                    <nm:maxAccess><read/></nm:maxAccess>
               </xs:appinfo>
       </xs:annotation>
       <xs:complexType>
             <xs:sequence>
                   <xs:element name="subscriber" type="xs:string">
                         <xs:annotation>
                             <xs:documentation>
                               This needs to be replaced with a more
                               prescriptive data type
                             </xs:documentation>
                         </xs:annotation>
                   </xs:element>

                   <xs:element name="namedProfile"
                           type="xs:string" minOccurs="0">
                         <xs:annotation>
                             <xs:documentation xml:lang="en">
                               The named profile associated with this
                               subscription. Note that the
                               contents of the named profile may have
                               changed since it was last applied
                             </xs:documentation>
                         </xs:annotation>
                         <xs:keyref refer="nsub:namedProfileKey"
                             name="namedProfileKeyRef">
                             <xs:selector xpath=".//namedProfile">
                              </xs:selector>
                             <xs:field xpath="namedProfile"></xs:field>
```

```
                          </xs:keyref>
                      </xs:element>

                      <xs:element name="status">
                            <xs:simpleType>
                              <xs:restriction base="xs:string">
                                <xs:enumeration value="Dormant"/>
                                <xs:enumeration value="Active"/>

                              </xs:restriction>
                            </xs:simpleType>

                          </xs:element>

                        </xs:sequence>
                      </xs:complexType>

                    </xs:element>

                  </xs:schema>
```

7.1.3.1.2 Modifications to Existing Operations

7.1.3.1.2.1   <create-subscription>

This capability adds a new attribute to the <create-subscription> command. This attribute is

callHome:

An optional parameter that, when present, indicates whether this will be a call-home Notification subscription. If not present, this will be a normal subscription.

7.1.3.1.3 Interactions with Other Capabilities

It is only when these subscriptions move from the dormant state to the active state that they have sessions associated with them. It is only at this point that they show up in the active subscription list.

8. Security Considerations

To be determined once specific aspects of this solution are better understood. In particular, the access control framework and the choice of transport will have a major impact on the security of the solution

9. IANA Considerations

Event Classes will likely be an IANA-managed resource. The initial set of values is defined in this specification.

10. Acknowledgements

Thanks to Gilbert Gagnon and Greg Wilbur for providing their input into the early work on this document. In addition, the editors would like to acknowledge input at the Vancouver editing session from the following people: Orly Nicklass, James Bakstrieve, Yoshifumi Atarashi, Glenn Waters, Alexander Clemm, Dave Harrington, Dave Partain, Ray Atarashi and Dave Perkins.

11. References

[NETCONF] Enns, R., "NETCONF Configuration Protocol", ID draft-ietf-netconf-prot-12, February 2006.

[NETCONF BEEP]
Lear, E. and K. Crozier, "Using the NETCONF Protocol over Blocks Extensible Exchange Protocol (BEEP)", ID draft-ietf-netconf-beep-10, March 2006.

[NETCONF Datamodel]
Chisholm, S. and S. Adwankar, "Framework for NETCONF Content", ID draft-chisholm-netconf-model-05.txt, April 2006.

[NETCONF SOAP]
Goddard, T., "Using the Network Configuration Protocol (NETCONF) Over the Simple Object Access Protocol (SOAP)", ID draft-ietf-netconf-soap-08, March 2006.

[NETCONF SSH]
Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure Shell (SSH)", ID draft-ietf-netconf-ssh-06.txt, March 2006.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C XML, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>.

[refs.RFC2026]
Bradner, S., "The Internet Standards Process -- Revision 3", RFC 2026, BCP 9, October 1996.

[refs.RFC2119]
Bradner, s., "Key words for RFCs to Indicate Requirements

Levels", RFC 2119, March 1997.

[refs.RFC2223]
Postel, J. and J. Reynolds, "Instructions to RFC Authors", RFC 2223, October 1997.

[refs.RFC3080]
Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

Authors' Addresses

Sharon Chisholm
Nortel
3500 Carling Ave
Nepean, Ontario K2H 8E9
Canada

Email: schishol@nortel.com

Kim Curran
Nortel
3500 Carling Ave
Nepean, Ontario K2H 8E9
Canada

Email: kicurran@nortel.com

Hector Trevino
Cisco
Suite 400
9155 E. Nichols Ave
Englewood, CO 80112
USA

Email: htrevino@cisco.com

Appendix A. Design Alternatives

A.1 Suspend And Resume

The purpose of the <cancel-subscription> operation is to stop event notification forwarding and since the notification subscription is transient the operation naturally removes all subscription configuration; For this reasons, a different mechanism might be needed for shutting down the notification session but preserving the subscription information thus allowing the NETCONF server to re-establish the parameters and reproduce the notification subscription.

The suspend and resume commands would allows a NETCONF client to suspend event notification forwarding without removing the existing subscription information. It could be used for both subscriptions based on persistent and non-persistent subscription information. Operations <suspend-subscription> and ><resume-subscription> are proposed for this purpose.

If event subscription information is now persistent, unsolicited session termination (i.e. other than <cancel-subscription)) is treated as if a <suspend-subscription> command was issued. Event forwarding is resumed by sending a <resume-subscription> to the NETCONF server on a new connection.

A.2 Lifecycle

Configuration information associated with the event subscription (event classes and filters) could persist beyond the life of the event subscription session. (i.e. it is maintained by the network element as part of its configuration). This configuration information is subject to the behaviour of the datastore it resides in and may or may not persist across re-boots (e.g. it could be part of the running configuration but not the startup configuration).

Appendix B. Event Notifications and Syslog

This appendix describes the mapping between syslog message fields and NETCONF event notification fields. The purpose of this mapping is to provide an unambiguous mapping to enable consistent multi-protocol implementations as well as to enable future migration.

The second part of the appendix describes an optional capability to embed an entire syslog message (hereafter referred to as syslog message(s) to avoid confusion with the message field in syslog) within a NETCONF event notification.

B.1 Leveraging Syslog Field Definitions

This section provides a semantic mapping between NETCONF event fields and syslog message fields.

```
  -----------------------------------------------------------------
  |         PRI          |            HEADER         |    MESSAGE      |
  -----------------------------------------------------------------
  | FACILITY | SEVERITY |  TIMESTAMP  | HOSTNAME  |  TAG CONTENT    |
  -----------------------------------------------------------------
```

Figure 2 - syslog message (RFC3164)

```
  -----------------------------------------------------------------
  |     HEADER          |    STRUCTURED DATA         |     MESSAGE     |
  -----------------------------------------------------------------
```

Figure 3 - syslog message (draft-ietf-syslog-protocol-14.txt)

HEADER (Version, Facility, Severity, Truncate, Flag, TimeStamp, HostName, AppName, ProcId, MsgId)

STRUCTURED DATA (Zero or more Structured Data Elements - SDEs)

MESSAGE (Text message)

B.1.1 Field Mapping

RFC3164	Syslog ID	NETCONF Event
VERSION		
FACILITY	FACILITY	
SEVERITY	SEVERITY	PerceivedSeverity
TRUNCATE FLAG		
TIMESTAMP	TIMESTAMP	EventTime
HOSTNAME	HOSTNAME	EventOrigin
TAG	APP-NAME	EventOrigin
PROC-ID		
MSG-ID		
CONTENT	CONTENT	AdditionalText

Figure 4 - syslog to NETCONF Event field mapping

Notes:

VERSION: Schema version is found in XML Schema namespace. However, no correspondence to syslog.

FACILITY: No well defined semantics for this field. Therefore not used at this time.

TRUNCATE: Not applicable. NETCONF events must be complete XML documents therefore cannot be truncated.

TIME: TIMESTAMP in syslog ID is derived from RFC3339 but with additional restrictions

PROC-ID: No equivalent field

CONTENT: This is a free form text field with not defined semantics. The contents of this field may be included in the AdditionalText field.

B.1.2 Severity Mapping

The severity value mappings stated in (draft-ietf-syslog-protocol-14) are used:

ITU Perceived Severity	syslog SEVERITY
Critical	Alert
Major	Critical
Minor	Error
Warning	Warning
Indeterminate	Notice
Cleared	Notice

Figure 5. ITU Perceived Severity to syslog SEVERITY mapping.

B.2 Syslog within NETCONF Events

B.2.1 Motivation

The syslog protocol (RFC3164) is widely used by equipment vendors as a means to deliver event messages. Due to the widespread use of syslog as well as a potential phased availability and coverage of NETCONF events by equipment vendors, it is envisioned that users will also follow a phased migration. As a way to facilitate migration and at the same time allow equipment vendors to provide comprehensive event coverage over a NETCONF event subscription session, syslog messages could be embedded in their entirety within the body of a NETCONF event notification.

The information provided in this appendix describes a mechanism to leverage syslog messages for the purpose of complementing the available NETCONF event notification set. The intent is to promote the use of the NETCONF interface and not to simply provide a wrapper and additional delivery mechanism for syslog messages. NETCONF events are intended to be well defined and structured, therefore providing an advantage over the unstructured and often times arbitrarily defined syslog messages (i.e. the message field).

Covered herein is the syslog protocol as defined in RFC3164 and draft-ietf-syslog-protocol-14.txt.

B.2.2 Embedding syslog messages in a NETCONF Event

When event notifications are supported, the default behaviour for a NETCONF server is to send NETCONF event notifications over an established event subscription. As an option, the NETCONF server may embed a syslog message in its entirety (e.g. RFC3164 - PRI, Header, and Message fields), placing it within the Event Info field

(SyslogInfo sub-field) - see Figure 1.

```
     ______________________________________________________
    | NETCONF Event  Header    |           Data           |
    |________________________ |__________________________|
    |                          |        Event Info        |
    |__________________________|__________________________|
                                             |
                                             v
                              ____________________________
                             | Event Fields | SyslogInfo |
                             |___________________________|
```

Figure 1 - Embedding syslog in a NETCONF Event Notifications

B.2.3 Supported Forwarding Options

Three event forwarding options may be supported by the NETCONF server: a) XML only (mandatory if NETCONF events capability is supported) b) XML and syslog (Optional) c) syslog only (optional)

Note to the reader: Option "a" above refers to event notification messages defined for use over the NETCONF protocol. While their use is not necessarily limited to NETCONF protocol, they are referred to as "NETCONF XML-event" in the remainder of this section simply to avoid ambiguity.

B.2.3.1 XML and Syslog option - Forwarding Behaviour

It is possible, due to coverage, for a given NETCONF implementation to not support a comprehensive set of NETCONF event notifications. Therefore, it is possible for a given event to trigger the generation of a syslog message without a NETCONF-aware counterpart. In such situations, the NETCONF server could form a NETCONF event notification, embed the syslog message in the SyslogInfo field and forward the NETCONF event notifications to all subscribed destinations. Otherwise, both NETCONF event and syslog messages must be included in the Event Info field.

B.2.3.2 Event Class Identification

The event class field is found in the NETCONF event header information as described in the main body of this document. It conveys information describing what type of event for which the event notification is generated and lets the consumer of the message know what sort of content to expect. NETCONF event notifications which

only contain a syslog message (Options c) must have the EventClass field set to "syslog". The NETCONF client parses the message in the same manner as any other message, finds the normal fields (ie, XML-marked content) not present and either proceeds to parse the SyslogInfo field or hands the syslog message to the entity responsible for processing syslog messages.

B.2.3.3 Event Subscription Options

A NETCONF client may request subscription to options b) XML and syslog or c) syslog only listed in "Supported Forwarding Options" at subscription time via the user-specified filter. The FILTER or NAMED FILTER parameter in <create-subscription>. As previously indicated, the default behaviour is to forward NETCONF XML only event notifications. [Editor's Note: How is this done exactly?]

B.2.3.4 Supported Forwarding Option Discovery

A potential means for a NETCONF server to convey its feature set support is via capabilities. However, in this particular case, the event content is not a protocol feature therefore other means are needed. A future version of this document will address this issue.

Appendix C. Example Configuration Notifications

This non-normative appendix provides a detailed description of a configuration change event notification definition in support of the configuration operations, particularly those defined by the NETCONF protocol.

C.1 Types of Configuration Events

Configuration event notifications include:

- o All-triggered Configuration Events

- o NETCONF-triggered Configuration Events

All-triggered Configuration events report on changes from the perspective of the managed resource, rather than the commands which created the configuration change. They are reported regardless of what specific method was used to initiate the change. They indicate that a change has occurred around hardware, software, services or other managed resources within a system. Specific events includes

- o Resource Added

- o Resource Removed

- o Resource Modified

NETCONF-triggered events are those which correspond to the execution of explicit NETCONF operations. These include:

- o copy-config event

  * This is a data store level event generated following the successful completion of a copy-config operation. This represents the creation of a new configuration file or replacement of an existing one.

- o delete-config event

  * This is a data store level event generated following the successful completion of a delete-config operation. This represents the deletion of a configuration file.

- o edit-config event

  * This is an event generated following a change in configuration due to an edit-config operation, e.g., due to the completion of

an edit-config operation which successfully changed some part of the configuration. See edit-config error-options (stop-on-error, ignore-error, rollback-on-error) The contents of this event are dependent on the type of operation performed: edit-config (merge, replace, delete, create). This event is not intended to report completely unsuccessful configuration operations.

- o lock-config event

  * This is a data store level event generated following the successful locking of a configuration data store.

- o unlock-config event

  * This is a data store level event generated following the successful release of a lock previously held on a configuration data store.

C.2 Config Event Notification Structure

The table below lists the EventInfo parameters for a config event notification.

Nomenclature:

O - This is marked optional field because it is implementation/ notification category dependent. In some cases this may be user configurable.

M - This is a mandatory field that must be included. Dependency on event class may exist as noted below

Parameter Name	Restrictions
EventInfo	
EventID	O
ResourceInstance	M
ConfigChangeType	M
TargetDataStore	M
UserInfo	O
UserName	
SourceIndicator	
TransactionId	
CopyConfigInfo	-- copy-config only
DataSource	M
EditConfigInfo	-- edit-config only
EventTime	M
Context	O
EnteredCommand	M
NewConfig	M
MergeReplaceInfo	
OldConfig	O
EventTime	M
EventGenerationTime	
EventSysUpTime	

C.3 Configuration Event Content

The applicability of these fields to other event classes is for further study.

C.3.1 Target Datastore

Target datastore refers to the data store (startup, candidate, running) which was modified by the management operation.

C.3.2 User Info

This is used to convey information describing who originated the configuration event and the means for submitting the request. The user info field contains the following information:

user Name: User id which was authorized to execute the associated management operation causing the generation of this event.

source Indicator: Indicates the method employed to initiate the management operation telnet, NETCONF, console, etc.

transaction Id: If available, this field contains a unique identifier for the associated management operation. This is implementation dependent and may require additional information to be communicated between server and client. A possible option is to make use of the message-id in the NETCONF rpc header

C.3.3 Data Source

The data source is used, for example, in the copy configuration command to indicated the source of information used in the copy operation

Applicable Event Classes: configuration (useful for copy-config)

C.3.4 Operation

Operation is used, for example, in the edit configuration command to indicated the specific operation that has taken place - create, delete, merge, replace.

Applicable Event Classes: configuration (useful for edit-config)

C.3.5 Context

The configuration sub-mode under which the command was executed.

Applicable Event Classes: configuration

C.3.6 Entered Command

The command entered and executed on the device.

C.3.7 New Config

The device's configuration following the successful execution of the entered command.

Applicable Event Classes: configuration

C.3.8 Old Config

The configuration prior to the execution of the entered command.

Applicable Event Classes: configuration

C.3.9 Non-netconf commands in configuration notifications

To support legacy implementations and for better integration with other deployed solutions on the box, sending information via netconf about configuration changes that were originated via other solutions, such as command line interfaces is necessary. In order to do this, the information in the message needs to be clearly tagged so that the consumer of the information knows what to expect. In addition, the creation of the subscription needs allow for the client to indicate whether this non-XML formatted information is of interest

The latter is done by identifying the XML namespace under which the data syntax/schema is defined. A NETCONF client requests the format in which it wants the NETCONF server to issue the event notifications at subscription time by specifying the appropriate namespace under the Filter parameter in the <create-subscription> operation. An example is provided below:

```
       <netconf:filter>
          <data-format:config-format-xml
                        xmlns="http://www.example.com/xmlnetevents"/>
       </netconf:filter>
```

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.