NETMOD
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2017

E. Voit
Cisco Systems
A. Bierman
YumaWorks
A. Clemm
Huawei
T. Jenkins
Cisco Systems

February 21, 2017

YANG Notification Headers and Bundles
draft-voit-netmod-yang-notifications2-00

Abstract

There are useful capabilities not available with existing YANG notifications as described in Section 7.16 of [RFC7950]. These include:

1. what are the set of transport agnostic header objects which might be usefully placed within YANG notifications.

2. how might a set of YANG notifications be bundled into a single transport message.

3. how do you query the originator of a notification to troubleshoot the bundling process

This specification provides technologies enabling these three capabilities.

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 August 25, 2017.

Table of Contents

1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [sn] and [yang-push]. Work on those documents has shown that additional capabilities in YANG notifications would be helpful. Three of these capabilities include:

1. what are the set of transport agnostic header objects which might be usefully placed within YANG notifications.

2. how might a set of YANG notifications be bundled into a single transport message.

3. how do you query the originator of a notification to troubleshoot the bundling process.

As none of these three capabilities are specific to subscriptions, it would be good to define them in a transport protocol agnostic way.

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].

Definitions of Notification, Event, Event Notification, Publisher, Receiver, and Subscription are defined in [sn].

3. Header Objects

There are a number of transport independent headers which should have common definition across applications. These include:

- record-type: what kind of information and have been assembled as part of this notification. (E.g., is it a YANG datastore update, an alarm, a syslog message, etc.)
- subscription-id: provides a reference into the reason the originator believed the receiver wishes to be notified of this specific information.
- record-severity: how important the originator feels this message to be.
- record-time: the time an event notification itself was recorded in the originating system.
- record-id: identifies an event notification on an originator.
- observation-domain-id: identifies the originator process which discovered and recorded the event notification. (note: look to reuse the domains set up with IPFIX.)
- notification-time: the time the message was packaged sent to the transport layer for delivery to the receiver.
- signature: allows an application to sign a message so that a receiver can verify the authenticity of the message.
- notification-id: identifies a message which includes one or more event records.

- previous-notification-id: the Notification id previously sent to a receiver. When used in conjunction with notification-id, this allow loss/duplication across previous messages to be discovered.

- message-generator-id: process which created the message notification. Allows identification of different line cards sending the notification messages. Used in conjunction with previous-notification-id, can help find drops and duplication when notifications are coming from multiple sources on a device. The logic is simple: if there is a message-generator-id in the header, then the previous-notification-id should been the notification-id the last time the message-generator-id was sent.

4. Headers added to an RFC7950 Notification

With the headers defined, they may now be applied to extend an RFC-7950 notification. This section provides examples of this.

The first thing which is done is to encapsulate these header fields within their own subtree in the notification message so that these objects can easily be decoupled, processed, and removed from any notification record payload.

It is useful to sequence these objects so that processing by the receiver is as efficient as possible, allowing the discarding of uninteresting notifications as quickly as possible. One record priority encoding would include the objects presented in the sequence above to help minimize event record processing CPU cycles. (Need to add more here, and acknowledge that different payloads and systems might benefit from alternative sequencing.)

```
    +---n notification-message
       +--ro notification-message-header
       |  +--ro record-time
       |  +--ro record-type?
       |  +--ro record-id?
       |  +--ro record-severity?
       |  +--ro observation-domain-id?
       |  +--ro subscription-id?
       |  +--ro notification-time?
       |  +--ro notification-id?
       |  +--ro previous-notification-id?
       |  +--ro signature?
       |  +--ro message-generator-id?
       +--ro receiver-record-contents?
```

An actual instance of a notification might look like:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
  <notification-message-header>
      <record-time>
          2017-02-14T00:00:02Z
      </record-time>
      <record-type>
          yang-patch
      </record-type>
      <subscription-identifier>
          823472
      </subscription-identifier>
      <notification-time>
          2017-02-14T00:00:05Z
      </notification-time>
      <notification-identifier>
          456
      </notification-identifier>
      <previous-notification-identifier>
          567
      </previous-notification-identifier>
      <signature>
          lKIo8s03fd23.....
      </signature>
  </notification-message-header>
  <datastore-changes>
      ...(yang patch here)...
  </datastore-changes>
</notification>
```

5. Bundled Notifications

In many implementations, it may be inefficient to transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

When this is done, one additional of a header field becomes valuable. This is the "record-count" which would tally the quantity of records which make up the contents of the bundle.

The format of a bundle would look as below. When compared to the unbundled notification, note that the headers have been split so that one set of headers associated with the notification occur once at the beginning of the message, and additional record specific headers which occur before individual records.

```
      +---n bundled-notification-message
         +--ro notification-message-header
         |  +--ro notification-time
         |  +--ro notification-id?
         |  +--ro previous-notification-id?
         |  +--ro signature?
         |  +--ro message-generator-id?
         |  +--ro record-count?
         +--ro notification-records*
            +--ro notification-record-header
            |  +--ro record-time
            |  +--ro record-type?
            |  +--ro record-id?
            |  +--ro record-severity?
            |  +--ro observation-domain-id?
            |  +--ro subscription-id?
            +--ro receiver-record-contents?
```

An actual instance of a bundled notification might look like:

```
     <notification
       xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
       <bundled-notification-message-header>
           <notification-time>
               2017-02-14T00:00:05Z
           </notification-time>
           <notification-identifier>
               456
           </notification-identifier>
           <previous-notification-identifier>
               567
           </previous-notification-identifier>
           <signature>
               lKIo8s03fd23...
           </signature>
           <record-count>
               2
           </record-count>
       </bundled-notification-message-header>
       <notification-record>
           <notification-record-header>
               <record-time>
                   2017-02-14T00:00:02Z
               </record-time>
               <record-type>
                   yang-patch
               </record-type>
               <subscription-identifier>
                   823472
               </subscription-identifier>
           </notification-record-header>
           <notification
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
              <datastore-changes>
                ...(yang patch here)...
              </datastore-changes>
           </notification>
       </notification-record>
       <notification-record>
               ...(record #2)...
       </notification-record>
     </notification>
```

6. Querying an Object Model

It is possible that an administrator would like to examine the contents of notifications via random access using a YANG model. There could be several values in such random access. These include:

- ability for applications to determine what message bundles were used to transport specific records.
- ability for applications to check which receivers have been sent an event notification.
- ability for applications to determine the time delta between event identification and transport.
- ability to reconstruct message passing during troubleshooting.
- ability to extract messages and records to evaluate whether the security filters have been properly applied.
- ability to compare the payloads of the same notification message sent to different receivers (again to evaluate the impact of the security filtering).

If such random access is needed, the YANG model structure below would enable random access to the information.

```
    +--ro notification-records
    |  +--ro notification-record* [record-id]
    |     +--ro record-time                    yang:date-and-time
    |     +--ro record-type              notification-record-format-type
    |     +--ro record-id                      uint32
    |     +--ro record-severity?               string
    |     +--ro observation-domain-id?         string
    |     +--ro notification-record-contents
    |     +--ro subscription-id*               subscription-ref
    +--ro notification-messages
    |  +--ro notification-message* [notification-id]
    |     +--ro notification-id                 uint32
    |     +--ro signature?                      string
    |     +--ro message-generator-id?           string
    |     +--ro notification-record             notification-record-ref
    |     +--ro receiver-notification-messages
    |        +--ro receiver-notification-message*
    |           +--ro receiver?                  receiver-ref
    |           +--ro notification-time          yang:date-and-time
    |           +--ro previous-notification-id?  uint32
    |           +--ro receiver-record-contents
    +--ro bundled-notification-messages
       +--ro bundled-notification-message* [notification-id]
          +--ro notification-id                   uint32
          +--ro signature?                        string
          +--ro message-generator-id?             string
          +--ro included-notification-records
          |  +--ro included-notification-record*
          |     +--ro notification-record?   notification-record-ref
          +--ro receiver-notification-messages
             +--ro receiver-notification-message*
                +--ro receiver?                      receiver-ref
                +--ro notification-time              yang:date-and-time
                +--ro previous-notification-id?      uint32
                +--ro record-count?                  uint16
                +--ro included-notification-records
                   +--ro notification-record*
                      +--ro receiver-record-contents
```

If such random access is not seen as needed, the model above should be discarded. This will also simplify the YANG model is the following section.

7. Data Model

```
 <CODE BEGINS> file "ietf-yang-notifications2.yang"
 module ietf-yang-notifications2 {
   yang-version 1.1;
```

```
   namespace "urn:ietf:params:xml:ns:yang:ietf-yang-notifications2";
   prefix yn2;

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

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

      WG Chair: Lou Berger
                <mailto:lberger@labn.net>

      WG Chair: Kent Watsen
                <mailto:kwatsen@juniper.net>

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

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

      Editor:   Tim Jenkins
                <mailto:timjenki@cisco.com>

      Editor:   Andy Bierman
                <mailto:andy@yumaworks.com>";


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

   revision 2017-02-23 {
     description
       "This module includes several definitions for two new yang
       notification message formats:
       (a) a message format including the definitions for common header
           information prior to notification payload.
       (b) a message format allowing the bundling of multiple
           notifications within it

       It also includes data nodes for querying related information such
```

```
       as:
          - ability to see contents of notifications both before and
            after any NACM filtering has been applied.
          - ability to see which message numbers have been sent to which
          receivers.";

     reference
       "draft-voit-netmod-yang-notifications2-00";
   }

   /*
    * IDENTITIES
    */

    /* Identities for notification record types */

    identity notification-record-format {
     description
       "Base identity to represent a different formats for notification
       records.";
   }

   identity system-event {
     base notification-record-format;
     description
       "System XML event";
   }

   identity yang-datastore {
     base notification-record-format;
     description
       "yang tree info";
   }

   identity yang-patch {
     base notification-record-format;
     description
       "yang patch record";
   }

   identity syslog-entry {
     base notification-record-format;
     description
       "Entry into syslog (figure linkage to existing draft.";
   }

   identity alarm {
     base notification-record-format;
```

```
    description
      "Alarm (perhaps link draft-sharma-netmod-fault-model-01 for more
      info)";
  }

  /*
   * TYPEDEFs
   */

  typedef notification-record-ref {
    type leafref {
      path "/notification-records/notification-record/record-id";
    }
    description
      "This type is used to reference a notification record (a.k.a.,
      event).";
  }

  typedef receiver-ref {
    type leafref {
      path "/sn:subscriptions/sn:subscription/sn:receivers/"+
      "sn:receiver/sn:address";
    }
    description
      "This type is used to reference a receiver.";
  }

  typedef subscription-ref {
    type leafref {
      path "/sn:subscriptions/sn:subscription/sn:identifier";
    }
    description
      "This type is used to reference a receiver.";
  }

  typedef notification-record-format-type {
    type identityref {
      base notification-record-format;
    }
    description
      "Type of notification record";
  }

  /*
   * GROUPINGS
   */

  grouping notification-message-header {
```

```
    description
      "Header information included with a notification.";
    leaf notification-id {
      type uint32;
      description
        "unique id for a notification which may go to one or many
        receivers.";
    }
    leaf signature {
      type string;
      description
        "Any originator signing of the contents of a notification
        message.  This can be useful for originating applications to
        verify record contents even when shipping over unsecure
        transport.";
    }
    leaf message-generator-id {
      type string;
      description
        "Software entity which created the notification message (e.g.,
        linecard 1).";
    }
  }

  grouping notification-message-receiver-header {
    description
      "Header information included with a notification which is
      specific to a receiver.";
    leaf notification-time {
      type yang:date-and-time;
      description
        "time the notification was generated prior to being sent to
        transport.";
    }
    leaf previous-notification-id {
      type uint32;
      description
        "Notification id previously sent by publisher to a specific
        receiver (allows detection of loss/duplication).";
    }
  }

  grouping notification-record-header {
    description
      "Common informational objects which might help a receiver
      interpret the meaning, details, and importance of an event
      notification.";
    leaf record-time {
```

```
      type yang:date-and-time;
      mandatory true;
      description
        "Time the system recognized the occurence of an event.";
    }
    leaf record-type {
      type notification-record-format-type;
      description
        "Describes the type of payload included.  This is turn allow
        the interpretation of the record contents.";
    }
    leaf record-id {
      type uint32;
      description
        "Identifier for the notification record.";
    }
    leaf record-severity {
      type string;
      description
        "System assigned severity.  (Likely we need to build/find an
        enumeration of common ones.)";
    }
    leaf observation-domain-id {
      type string;
      description
        "Software entity which created the notification record (e.g.,
        process id).";
    }
  }

  grouping subscribed-notification-record-header {
    description
      "Header information included with a notification.";
    uses notification-record-header;
    leaf subscription-id {
      type uint32;
      description
        "Id of the subscription which led to the notification being
        generated.";
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification notification-message {
    description
```

```
      "Notification message to a receiver containing only one event.";
    container notification-message-header {
      description
        "delineates header info from notification messages for easy
        parsing.";
        uses subscribed-notification-record-header;
        uses notification-message-header;
        uses notification-message-receiver-header;
    }
    anydata receiver-record-contents {
      description
        "Non-header info of what actually got sent to receiver after
        security filter.  (Note: Possible to have extra process
        encryption.)";
    }
  }

  notification bundled-notification-message {
    description
      "Notification message to a receiver containing many events,
      possibly relating to independent subscriptions.";
    container bundled-notification-message-header {
        description
            "Delineates header info from notification messages for easy
            parsing.";
        uses notification-message-header;
        uses notification-message-receiver-header {
          refine notification-time {
            mandatory true;
          }
        }
        leaf record-count {
            type uint16;
            description
                "Quantity of events in a bundled-notification-message
                for a specific receiver.  This value is per receiver in
                case an entire notification record is filtered out.";
        }
    }

    list notification-records {
      description
        "Set of messages within a notification to a receiver.";
      container notification-record-header {
        description
          "delineates header info from notification messages for easy
          parsing.";
        uses subscribed-notification-record-header;
```

```
      }
      anydata receiver-record-contents {
        description
          "Non-header info of what actually got sent to receiver after
          security filter.  (Note: Possible to have extra process
          encryption.)";

      }
    }
  }

  /*
   * DATA NODES
   */

  container notification-records {
    config false;
    description
      "Maintains instances of event notifications recorded by the
      system.";
    list notification-record {
      key "record-id";
      description
        "Specific instances of event notifications recorded by the
        system.";
      uses notification-record-header {
        refine record-id {
          mandatory true;
        }
        refine record-type {
          mandatory true;
        }
      }
      anydata notification-record-contents {
        mandatory true;
        description
          "Notification event contents independent of any receiver
          security filtering.";
      }
      leaf-list subscription-id {
        type subscription-ref;
        description
          "Instances of subscriptions which should receive or have
          received this event notification record.";
      }
    }
  }
  container notification-messages {
```

```
    config false;
    description
      "Contains a history of the notification messages which have been
      generated.";
    list notification-message {
      key "notification-id";
      description
        "Instances of notification messages generated with the intent
        of sending them to one or more receivers.";
      uses notification-message-header {
        refine notification-id {
          mandatory true;
        }
      }
      leaf notification-record {
        type notification-record-ref;
        mandatory true;
        description
          "Included notification. The record itself, or elements of
          this record might not be sent to any included receiver based
          on security permissions for that receiver.";
      }
      container receiver-notification-messages {
        description
          "Contains a history of messages targeted for a receiver.";
         list receiver-notification-message {
          description
            "Maintains instances of messages targeted for a receiver.";
          leaf receiver {
            type receiver-ref;
            description
              "Reference to the recipient targeted for this
              notification message.  (This also allows the unique
              identification of the subscription.)";
          }
          uses notification-message-receiver-header {
            refine notification-time {
              mandatory true;
            }
          }
          anydata receiver-record-contents {
            mandatory true;
            description
              "The specific security filtered contents of one record
              going to a receiver.";
          }
        }
      }
```

```
    }
  }
  container bundled-notification-messages {
    config false;
    description
      "Contains a history of bundled notification messages which have
      been generated.";
    list bundled-notification-message {
      key "notification-id";
      min-elements 1;
      description
        "Maintains instances of a bundled notification messages
        generated with the intent of sending them to one or more
        receivers.";
      uses notification-message-header{
        refine notification-id {
          mandatory true;
        }
      }
      container included-notification-records {
          description
          "Contains specific records included in the bundle.";
        list included-notification-record {
          description
            "A specific instance of record included in a bundle.";
          leaf notification-record {
            type notification-record-ref;
            description
              "Included notification within the bundle.  Full records
              or elements of this record might not be sent to any
              included receiver based on security permissions for that
              receiver.";
          }
        }
      }
      container receiver-notification-messages {
          description
          "Contains instances of messages generated for a specific
          receiver.";
         list receiver-notification-message {
          description
            "Maintains instances of bundled messages targeted for a
            receiver.";
           leaf receiver {
            type receiver-ref;
            description
              "Reference to the recipient targeted for this bundled
              notification message.  (As a receiver is unique to a
```

```
                subscription, this also identifies the subscription
                explicitly.  If something other than receiver is used, a
                method to identify the subscription is also needed as it
                can't automatically be derived from the notification
                record.";
            }
            uses notification-message-receiver-header {
              refine notification-time {
                mandatory true;
              }
            }
            leaf record-count {
              type uint16;
                description
                "Number of records actually sent to a receiver after
                considering the application of NACM policies on the
                notification records.";
            }
            container included-notification-records {
              description
                "Contains the records sent to a receiver within a
                specific notification message.";
              list notification-record {
                description
                  "Maintains instances of records sent to a receiver.";
                anydata receiver-record-contents {
                  mandatory true;
                    description
                    "The specific security filtered contents of one
                    record going to a receiver.";
                }
              }
            }
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

8. Security Considerations

to be populated

9. References

9.1. 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>.

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

9.2. Informative References

[sn] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to Event Notifications", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

[yang-push] Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. Issues being worked

(To be removed by RFC editor prior to publication)

We need to define the ways to invoke and configure the capability within an originating device. This includes defining what header types are selected.

Should we allow multiple subscriptions to be associated with an update record via a leaf-list?

Should the subscription id in a notification actually be a leafref?

We need to do a lot more to discuss transport efficiency implications.

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Tim Jenkins
Cisco Systems

Email: timjenki@cisco.com