NETCONF
Internet-Draft
Intended status: Standards Track
Expires: October 27, 2017

E. Voit
Cisco Systems
A. Bierman
YumaWorks
A. Clemm
Huawei
T. Jenkins
Cisco Systems
April 25, 2017

Notification Message Headers and Bundles
draft-voit-netconf-notification-messages-00

Abstract

This document specifies transport independent capabilities for messages transporting event notifications and YANG datastore update records. Included are:

- a set of transport agnostic message header objects, and
- how to associate a subset of these header objects with one or more events, YANG datastore updates, and/or alarms.

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 October 27, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [subscribe] and [yang-push]. Work on those documents have shown that existing YANG notifications described in [RFC7950] section 7.16 do not expose some useful transport independent capabilities that application developers are requesting. Communicating information on the following objects should not require knowledge of the underlying transport:

o the kind of information encapsulated (event, data objects, alarm?)

o the time information was generated

o the time the information was sent

o a signature to verify authenticity

o the process generating the information

o an originating request correlation

- an ability to bundle information records together in a message

- the ability to check for message loss/reordering

The document describes information elements needed for the functions above. It also provides YANG Notifications for sending messages containing bone or more events and/or update records to a receiver.

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, Receiver, and Subscription are defined in [subscribe].

3. Header Objects

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

- record-type: defines the kind of information assembled as a unit. (E.g., is it a YANG datastore update, an alarm, an event, etc.)

- subscription-id: provides a reference into the reason the originator believed the receiver wishes to be notified of this specific information.

- record-time: the time an event, datastore update, or other item it itself is recognized 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.

- dscp: network qos encoding which an application suggests should be applied to the message.

- notification-id: for a specific message generator, this identifies a message which includes one or more event records.

- previous-notification-id: the notification-id of the message preceding the current one intended for the same receiver. When used in conjunction with the current notification-id, this allows loss/duplication across previous messages to be discovered.

- message-generator-id: identifier for the process which created the message notification. This allows disambiguation of an information source, such as the identification of different line cards sending the notification messages. Used in conjunction with previous-notification-id, this can help find drops and duplications when notifications are coming from multiple sources on a device. If there is a message-generator-id in the header, then the previous-notification-id should be the notification-id from the last time that message-generator-id was sent.

4. Transport independent headers for notifications

Many objects may be placed within a notification. However only a certain subset these objects are of potential use to networking layers prior the notification being interpreted by some receiving application layer process. By exposing this object information as part of a header, and by using standardized object names, it becomes possible for this object information to be leveraged in transit.

The objects defined in the previous section effectively become well-known objects where, if in the header, may act as supplemental information in communications between two devices. These well-known header fields are encapsulated within a dedicated subtree which leads off the notification message. This allows header objects to be easily be decoupled, stripped, and processed separately.

Typically sequence of information in YANG models is irrelevant. But as part of a transported notification, It is useful to sequence these header objects so that processing is as efficient as possible. This allows the handling or discarding of uninteresting notifications quickly.

Below is are record objects contents would include the objects presented in the section above. The proper way this message would be generated would be to look for the well known object names and place them in the header. All other would be placed in the notification record contents. (Note: are there any of these we should rather duplicate than move?)

```
      +---n notification-message
         +--ro notification-message-header
         |  +--ro record-time                 yang:date-and-time
         |  +--ro record-type?                notification-record-format
         |  +--ro subscription-id*            uint32
         |  +--ro record-id?                  uint32
         |  +--ro observation-domain-id?      string
         |  +--ro notification-id?            uint32
         |  +--ro notification-time?          yang:date-and-time
         |  +--ro previous-notification-id?   uint32
         |  +--ro dscp?                       inet:dscp
         |  +--ro message-generator-id?       string
         |  +--ro signature?                  string
         +--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 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 bundled-notification-message-header
   |  +--ro notification-id?           uint32
   |  +--ro notification-time          yang:date-and-time
   |  +--ro previous-notification-id?  uint32
   |  +--ro dscp?                      inet:dscp
   |  +--ro message-generator-id?      string
   |  +--ro signature?                 string
   |  +--ro record-count?              uint16
   +--ro notification-records*
      +--ro notification-record-header
      |  +--ro record-time             yang:date-and-time
      |  +--ro record-type?            notification-record-format
      |  +--ro subscription-id*        uint32
      |  +--ro record-id?              uint32
      |  +--ro observation-domain-id?  string
      +--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. Data Model

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

```
   namespace "urn:ietf:params:xml:ns:yang:ietf-notification-messages";
   prefix nm;

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

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

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

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

      Editor:   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
     notification messages with well known header objects.";

   revision 2017-04-25 {
     description
       "This module includes definitions for two new yang
       notification message objects:
       (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";

     reference
```

```
       "draft-voit-netconf-notification-messages-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 data node / tree extract";
   }
   identity yang-patch {
     base notification-record-format;
     description
       "yang patch record";
   }
   identity syslog-entry {
     base notification-record-format;
     description
       "Unstructured syslog entry.";
   }
   identity alarm {
     base notification-record-format;
     description
       "Alarm (perhaps link draft-sharma-netmod-fault-model-01 for more
       info)";
   }

   /*
    * TYPEDEFs
    */

   typedef notification-record-format {
     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 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).";
     }
     leaf dscp {
       type inet:dscp;
       default "0";
       description
         "The push update's IP packet transport priority. This is made
         visible across network hops to receiver. The transport
         priority is shared for all receivers of a given
         subscription.";
     }
     leaf message-generator-id {
       type string;
       description
         "Software entity which created the notification message (e.g.,
         linecard 1).";
     }
     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.";
      }
    }

    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 occurrence of an event.";
      }
      leaf record-type {
        type notification-record-format;
        description
          "Describes the type of payload included.  This is turn allow
          the interpretation of the record contents.";
      }
      leaf-list subscription-id {
        type uint32;
        description
          "Id of the subscription which led to the notification being
          generated.";
      }
      leaf record-id {
        type uint32;
        description
          "Identifier for the notification record.";
      }
      leaf observation-domain-id {
        type string;
        description
          "Software entity which created the notification record (e.g.,
          process id).";
      }
    }

    /*
     * 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 notification-record-header;
       uses notification-message-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 {
           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 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.)";

      }
    }
  }
}
<CODE ENDS>
```

7. Security Considerations

More needs to be thought through here, as this adds additional information onto notifications, the security implications shouldn't be singificantly beyond those from [subscribe] other than ensuring that data plane devices can accomplish the necessary content filtering despite the potential of a new level of header being applied.

8. References

8.1. 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>.

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

8.2. Informative References

[initial-version]
Voit, E., Bierman, A., Clemm, A., and T. Jenkins, "Custom Subscription to Event Notifications", February 2017, <https://tools.ietf.org/html/draft-voit-netmod-yang-notifications2-00>.

[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>.

[yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", April 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. This also includes knowning the header types which can be supported by a receiver.

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

Relationship with DTN protocols.

Appendix B. Querying an Object Model

(This appendix was in a previous iteration of this draft. It has been removed as unlikely to be seen in an implementation. It is being kept in for discussion purposes.)

It is also possible that that external entities might want to query message information after it has been sent. And therefore it is possible that an administrator would like to examine the contents of notifications via random access using a YANG model rather that Syslog. 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, it is possible to extend the YANG model data model document to enable random access to the information. A cut at what this might look like can be seen in [initial-version]

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