Abstract

This document defines a new notification message format. Included are:

- a new notification mechanism and encoding to replace the one way operation of RFC-5277
- a set of common, transport agnostic message header objects.
- how to bundle multiple event records into a single notification message.
- how to ensure these new capabilities are only used with capable receivers.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on February 16, 2020.
1. Introduction

Mechanisms to support subscription to event notifications have been
defined in [I-D.ietf-netconf-subscribed-notifications] and
[I-D.ietf-netconf-yang-push]. Work on those documents has shown that
notifications described in [RFC7950] section 7.16 could benefit from
transport independent headers. With such headers, communicating the
following information to receiving applications can be done without explicit linkage to an underlying transport protocol:

- the time the notification was generated
- the time the notification was placed in a message and queued for transport
- an identifier for the process generating the notification
- signatures to verify authenticity
- a subscription id which allows a notification be correlated with a request for that notification
- multiple notifications bundled into one transportable message
- a message-id allowing a receiver to check for message loss/reordering

The document describes information elements needed for the functions above. It also provides instances of YANG structures [I-D.draft-ietf-netmod-yang-data-ext] for sending messages containing one or more notifications to a receiver.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The definition of notification is in [RFC7950] Section 4.2.10. Publisher, receiver, subscription, and event occurrence time are defined in [I-D.draft-ietf-netconf-subscribed-notifications].

3. Header Objects

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

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

- notification-time: the origination time where a notification was fully generated within the publisher.
4. Encapsulation of Header Objects in Messages

A specific set of well-known objects are of potential use to networking layers prior 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 are these well-known header objects. These objects are identified within a dedicated header subtree which leads off a particular transportable message. This allows header objects to be easily decoupled, stripped, and processed separately.

A receiver which supporting this document MUST be able to handle receipt of either type of message from a publisher.

4.1. One Notification per Message

This section has been deleted from previous versions. It will be re-instated if NETCONF WG members are not comfortable with the
efficiency of the solution which can encode many notifications per message, as described below.

4.2. Many Notifications per Message

While possible in some scenarios, it often inefficient to marshal and transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

The format of this bundle appears in the YANG structure below, and is fully defined in the YANG module. There are three parts of this bundle:

- a message header describing the marshaling, including information such as when the marshaling occurred
- a list of encapsulated information
- an optional message footer for whole-message signing and message-generator integrity verification.

Within the list of encapsulated notifications, there are also three parts:

- a notification header defining what is in an encapsulated notification
- the actual notification itself
- an optional notification footer for individual notification signing and observation-domain integrity verification.
structure message
  +--ro message!
      +--ro message-header
        |   +--ro message-time               yang:date-and-time
        |   +--ro message-id?                uint32
        |   +--ro message-generator-id?      string
        |   +--ro notification-count?        uint16
        +--ro notifications*
            +--ro notification-header
              |   +--ro notification-time         yang:date-and-time
              |   +--ro yang-module?              yang:yang-identifier
              |   +--ro subscription-id*          uint32
              |   +--ro notification-id?          uint32
              |   +--ro observation-domain-id?    string
              +--ro notification-contents?
              +--ro notification-footer!
                +--ro signature-algorithm    string
                +--ro signature-value        string
                +--ro integrity-evidence?    string
          +--ro message-footer!
            +--ro signature-algorithm    string
            +--ro signature-value        string
            +--ro integrity-evidence?    string

An XML instance of a message might look like:
<structure bundled-message
xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0">
<message-header>
<message-time>
2017-02-14T00:00:05Z
</message-time>
<message-id>
456
</message-id>
<notification-count>
2
</notification-count>
</message-header>
<notifications>
<notification>
<notification-header>
<notification-time>
2017-02-14T00:00:02Z
</notification-time>
<subscription-id>
823472
</subscription-id>
<yang-module>
ietf-yang-push
</yang-module>
<yang-notification-name>
push-change-update
</yang-notification-name>
</notification-header>
<notification-contents>
<datastore-changes-xml>
<alpha xmlns="http://example.com/sample-data/1.0">
<beta urn:ietf:params:xml:ns:netconf:base:1.0:
operation="delete"/>
</alpha>
</datastore-changes-xml>
</push-change-update>
</notification-contents>
</notification>
...(record #2)...
</notifications>
</structure>
5. Configuration of Headers

A publisher MUST select the set of headers to use within any particular message. The two mandatory headers which MUST always be applied are 'message-time' and 'subscription-id'.

Beyond these two mandatory headers, additional headers MAY be included. Configuration of what these optional headers should be can come from the following sources:

1. Publisher wide default headers which are placed on all notifications. An optional header is a publisher default if its identity is included within the 'additional-headers' leaf-list.

2. More notification specific headers may also be desired. If new headers are needed for a specific type of YANG notification, these can be populated through 'additional-notification-headers' leaf-list.

3. An application process may also identify common headers to use when transporting notifications for a specific subscription. How such application specific configuration is accomplished within the publisher is out-of-scope.

The set of headers selected and populated for any particular message is derived from the union of the mandatory headers and configured optional headers.

The YANG tree showing elements of configuration is depicted in the following figure.

```
module: ietf-notification-messages
    +--rw additional-default-headers {publisher)?
        +--rw additional-headers* optional-header
        +--rw yang-notification-specific-default* [yang-module yang-notification-name]
            +--rw yang-module yang:yang-identifier
            +--rw yang-notification-name notification-type
            +--rw additional-notification-headers* optional-notification-header

Configuration Model structure
```

Of note in this tree is the optional feature of 'publisher'. This feature indicates an ability to send notifications. A publisher supporting this specification MUST also be able to parse any messages received as defined in this document.
6. Discovering Receiver Support

We need capability exchange from the receiver to the publisher at transport session initiation to indicate support for this specification.

For all types of transport connections, if the receiver indicates support for this specification, then it MAY be used. In addition, [RFC5277] one-way notifications MUST NOT be used if the receiver indicates support for this specification to a publisher which also supports it.

Where NETCONF transport is used, advertising this specification’s namespace during an earlier client capabilities discovery phase MAY be used to indicate support for this specification:

```xml
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>
            urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0
        </capability>
    </capabilities>
    <session-id>4</session-id>
</hello>
```

NOTE: It is understood that even though it is allowed in [RFC6241] section 8.1, robust NETCONF client driven capabilities exchange is not something which is common in implementation. Therefore reviewers are asked to submit alternative proposals to the mailing list.

For RESTCONF, a mechanism for capability discovery is TBD. Proposals are welcome here.

The mechanism described above assumes that a capability discovery phase happens before a subscription is started. This is not always the case. There is no guarantee that a capability exchange has taken place before the messages are emitted. A solution for this in the case of HTTP based transport could be that a receiver would have to reply "ok" and also return the client capabilities as part a response to the initiation of the POST.

7. YANG Module

```yaml
<CODE BEGINS> file "ietf-notification-messages@2019-08-16.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-yang-structure-ext { prefix sx; }

organization "IETF";
contact
  "WG Web:  <http://tools.ietf.org/wg/netconf/>
  WG List:  <mailto:netconf@ietf.org>
  Editor:   Eric Voit
            <mailto:evoit@cisco.com>
  Editor:   Henk Birkholz
            <mailto:henk.birkholz@sit.fraunhofer.de>
  Editor:   Alexander Clemm
            <mailto:ludwig@clemm.org>
  Editor:   Andy Bierman
            <mailto:andy@yumaworks.com>
  Editor:   Tim Jenkins
            <mailto:timjenki@cisco.com">

description
  "This module contains conceptual YANG specifications for
  messages carrying notifications with well-known header objects.";

revision 2019-08-16 {
  description
    "Initial version."
  reference
    "draft-ietf-netconf-notification-messages-07"
}

/*
 * FEATURES
 */

feature publisher {
  description
    "This feature indicates that support for both publisher and
    receiver of messages complying to the specification.";
}
Identities for common headers

identity common-header {
  description
    "A well-known header which can be included somewhere within a
    message.";
}

description message-time {
  base common-header;
  description
    "Header information consisting of time the message headers were
generated prior to being sent to transport";
}

description subscription-id {
  base common-header;
  description
    "Header information consisting of the identifier of the
subscription associated with the notification being
encapsulated.";
}

description notification-count {
  base common-header;
  description
    "Header information consisting of the quantity of notifications in
a bundled-message for a specific receiver.";
}

description optional-header {
  base common-header;
  description
    "A well-known header which an application may choose to include
within a message.";
}

description message-id {
  base optional-header;
  description
    "Header information that identifies a message to a specific
receiver";
}
identity message-generator-id {
  base optional-header;
  description
    "Header information consisting of an identifier for a software entity which created the message (e.g., linecard 1).";
}

identity message-signature {
  base optional-header;
  description
    "Identifies two elements of header information consisting of a signature and the signature type for the contents of a message. Signatures can be useful for originating applications to verify record contents even when shipping over unsecure transport.";
}

identity message-integrity-evidence {
  base optional-header;
  description
    "Header information consisting of the information which backs up the assertions made as to the validity of the information provided within the message.";
}

identity optional-notification-header {
  base optional-header;
  description
    "A well-known header which an application may choose to include within a message.";
}

identity notification-time {
  base optional-notification-header;
  description
    "Header information consisting of the time an originating process created the notification.";
}

identity notification-id {
  base optional-notification-header;
  description
    "Header information consisting of an identifier for an instance of a notification. If access control based on a message's receiver may strip information from within the notification, this notification-id MUST allow the identification of the specific contents of notification as it exits the publisher.";
}
identity observation-domain-id {
  base optional-notification-header;
  description
    "Header information identifying the software entity which created
    the notification (e.g., process id).";
}

identity notification-signature {
  base optional-notification-header;
  description
    "Header information consisting of a signature which can be used to prove
    the authenticity that some asserter validates over the information
    provided within the notification.";
}

identity notification-integrity-evidence {
  base optional-notification-header;
  description
    "Header information consisting of the information which backs up
    the assertions made as to the validity of the information
    provided within the notification.";
}

 /*************************************************************************
  * TYPEDEFs
  */

typedef optional-header {
  type identityref {
    base optional-header;
  }
  description
    "Type of header object which may be included somewhere within a
    message.";
}

typedef optional-notification-header {
  type identityref {
    base optional-notification-header;
  }
  description
    "Type of header object which may be included somewhere within a
    message.";
}
typedef notification-type {
    type string {
        pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
    }
    description
    "The name of a notification within a YANG module.";
    reference
    "RFC-7950 Section 7.16";
}

/*
 * GROUPINGS
 */

grouping message-header {
    description
    "Header information included with a message.";
    leaf message-time {
        type yang:date-and-time;
        mandatory true;
        description
        "Time the message was generated prior to being sent to
         transport.";
    }
    leaf message-id {
        type uint32;
        description
        "Id for a message going to a receiver from a message
generator. The id will increment by one with each message sent
from a particular message generator, allowing the message-id
to be used as a sequence number.";
    }
    leaf message-generator-id {
        type string;
        description
        "Software entity which created the message (e.g., linecard 1).
The combination of message-id and message-generator-id must be
unique until reset or a roll-over occurs.";
    }
    leaf notification-count {
        type uint16;
        description
        "Quantity of notifications in a bundled-message to a
specific receiver.";
    }
}

grouping notification-header {

description
"Common informational objects which might help a receiver interpret the meaning, details, or importance of a notification."
leaf notification-time {
    type yang:date-and-time;
    mandatory true;
    description
        "Time the system recognized the occurrence of an event.";
}
leaf yang-module {
    type yang:yang-identifier;
    description
        "Name of the YANG module supported by the publisher.";
}
leaf-list subscription-id {
    type uint32;
    description
        "Id of the subscription which led to the notification being generated.";
}
leaf notification-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).";
}
}

grouping security-footer {
    description
        "Reusable grouping for common objects which apply to the signing of notifications or messages.";
    leaf signature-algorithm {
        type string;
        mandatory true;
        description
            "The technology with which an originator signed of some delineated contents.";
    }
    leaf signature-value {
        type string;
        mandatory true;
        description
        
    }
}
"Any originator signing of the contents of a header and content. This is useful for verifying contents even when shipping over unsecure transport.";

leaf integrity-evidence {
  type string;
  description
  "This mechanism allows a verifier to ensure that the use of the private key, represented by the corresponding public key certificate, was performed with a TCG compliant TPM environment. This evidence is never included in within any signature."
  reference
  "TCG Infrastructure Workgroup, Subject Key Attestation Evidence Extension, Specification Version 1.0, Revision 7.";
}

/*
* YANG encoded structures which can be sent to receivers
*/

sx:structure message {
  container message {
    presence
    "Indicates attempt to communicate notifications to a receiver.";
    description
    "Message to a receiver containing one or more notifications";
    container message-header {
      description
      "Header info for messages.";
      uses message-header;
    }
    list notifications {
      description
      "Set of notifications to a receiver.";
      container notification-header {
        description
        "Header info for a notification.";
        uses notification-header;
      }
      anydata notification-contents {
        description
        "Encapsulates objects following YANG’s notification-stmt grammar of RFC-7950 section 14. Within are the notified objects the publisher actually generated in order to be passed to a receiver after all filtering has completed."
      }
    }
  }
}
container notification-footer {
    presence
    "Indicates attempt to secure a notification.";
    description
    "Signature and evidence for messages.";
    uses security-footer;
}
}
container message-footer {
    presence
    "Indicates attempt to secure the entire message.";
    description
    "Signature and evidence for messages.";
    uses security-footer;
}
}
*/
* DATA-NODES
*/

container additional-default-headers {
    if-feature "publisher";
    description
    "This container maintains a list of which additional notifications
    should use which optional headers if the receiver supports this
    specification.";
    leaf-list additional-headers {
        type optional-header;
        description
        "This list contains the identities of the optional header types
        which are to be included within each message from this
        publisher.";
    }
    list yang-notification-specific-default {
        key "yang-module yang-notification-name";
        description
        "For any included YANG notifications, this list provides
        additional optional headers which should be placed within the
        container notification-header if the receiver supports this
        specification. This list incrementally adds to any headers
        indicated within the leaf-list 'additional-headers'.";
        leaf yang-module {
            type yang:yang-identifier;
            description
            "Name of the YANG module supported by the publisher.";
        }
    }
}
leaf yang-notification-name {
    type notification-type;
    description
        "The name of a notification within a YANG module."
}
leaf-list additional-notification-headers {
    type optional-notification-header;
    description
        "The set of additional default headers which will be sent
         for a specific notification."
}

8. Backwards Compatibility

With this specification, there is no change to YANG’s ‘notification’ statement

Legacy clients are unaffected, and existing users of [RFC5277],
[RFC7950], and [RFC8040] are free to use current behaviors until all
involved device support this specification.

9. Security Considerations

Certain headers might be computationally complex for a publisher to deliver. Signatures or encryption are two examples of this. It MUST be possible to suspend or terminate a subscription due to lack of resources based on this reason.

Decisions on whether to bundle or not to a receiver are fully under the purview of the Publisher. A receiver could slow delivery to existing subscriptions by creating new ones. (Which would result in the publisher going into a bundling mode.)

10. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Martin Bjorklund, Einar Nilsen-Nygaard, and Kent Watsen.

11. References
11.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]

[I-D.draft-ietf-netmod-yang-data-ext]


11.2. Informative References

[I-D.draft-ietf-netconf-restconf-notif]

[I-D.ietf-netconf-yang-push]

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v06 - v07

- Updated author affiliation.

v05 - v06

- With SN and YP getting RFC numbers, revisiting this document.
- Changed yang-data to draft-ietf-netmod-yang-data-ext’s ‘structure’.
- Removed the ability to reference structures other than YANG notifications.

v04 - v05

- Revision before expiration. Awaiting closure of SN and YP prior to update.

v03 - v04

- Terminology tweaks.
- Revision before expiration. Awaiting closure of SN prior to update.

v02 - v03

- Removed the option for an unbundled message. This might be re-added later for transport efficiency if desired by the WG.
- New message structure driven by the desire to put the signature information at the end.

v01 - v02
- Fixed the yang-data encapsulation container issue
- Updated object definitions to point to RFC-7950 definitions
- Added headers for module and notification-type.

**v00 - v01**

- Alternative to 5277 one-way notification added
- Storage of default headers by notification type
- Backwards compatibility
- Capability discovery
- Move to yang-data

- Removed dscp and record-type as common headers. (Record type can be determined by the namespace of the record contents. Dscp is useful where applications need internal communications within a Publisher, but it is unclear as to whether this use case need be exposed to a receiver.

### Appendix B. Issues being worked

(To be removed by RFC editor prior to publication)

A complete JSON document is supposed to be sent as part of Media Type "application/yang-data+json". As we are sending separate notifications after each other, we need to choose whether we start with some extra encapsulation for the very first message pushed, or if we want a new Media Type for streaming updates.

Improved discovery mechanisms for NETCONF

Need to ensure the proper references exist to a notification definition driven by RFC-7950 which is acceptable to other eventual users of this specification.

### Appendix C. Subscription Specific Headers

(To be removed by RFC editor prior to publication)

This section discusses a future functional addition which could leverage this draft. It is included for informational purposes only.
A dynamic subscriber might want to mandate that certain headers be used for push updates from a publisher. Some examples of this include a subscriber requesting to:

- establish this subscription, but just if transport messages containing the pushed data will be encrypted,
- establish this subscription, but only if you can attest to the information being delivered in requested notification records, or
- provide a sequence-id for all messages to this receiver (in order to check for loss).

Providing this type of functionality would necessitate a new revision of the [I-D.draft-ietf-netconf-subscribed-notifications]’s RPCs and state change notifications. Subscription specific header information would overwrite the default headers identified in this document.

Appendix D. Implications to Existing RFCs

(To be removed by RFC editor prior to publication)

YANG one-way exchanges currently use a non-extensible header and encoding defined in section 4 of RFC-5277. These RFCs MUST be updated to enable this draft. These RFCs SHOULD be updated to provide examples

D.1. Implications to RFC-7950

Sections which expose netconf:capability:notification:1.0 are 4.2.10

Sections which provide examples using netconf:notification:1.0 are 7.10.4, 7.16.3, and 9.9.6

D.2. Implications to RFC-8040

Section 6.4 demands use of RFC-5277’s netconf:notification:1.0, and later in the section provides an example.

Authors’ Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com
Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Alexander Clemm
Futurewei

Email: ludwig@clemm.org

Tim Jenkins
Cisco Systems

Email: timjenki@cisco.com