Network Working Group   R. Enns, Ed.
Internet-Draft   Juniper Networks
Obsoletes: 4741 (if approved)   M. Bjorklund, Ed.
Intended status: Standards Track   Tail-f Systems
Expires: September 15, 2011   J. Schoenwaelder, Ed.
                                                     Jacobs University
                                                      A. Bierman, Ed.
                                                              Brocade
                                                       March 14, 2011

                   Network Configuration Protocol (NETCONF)
                         draft-ietf-netconf-4741bis-10

   Abstract

   The Network Configuration Protocol (NETCONF) defined in this document
   provides mechanisms to install, manipulate, and delete the
   configuration of network devices. It uses an Extensible Markup
   Language (XML)-based data encoding for the configuration data as well
   as the protocol messages. The NETCONF protocol operations are
   realized as Remote Procedure Calls (RPC). This document obsoletes
   RFC 4741.

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 September 15, 2011.

Table of Contents

1. Introduction

   The NETCONF protocol defines a simple mechanism through which a
   network device can be managed, configuration data information can be
   retrieved, and new configuration data can be uploaded and
   manipulated. The protocol allows the device to expose a full, formal
   application programming interface (API). Applications can use this
   straightforward API to send and receive full and partial
   configuration data sets.

   The NETCONF protocol uses a remote procedure call (RPC) paradigm. A
   client encodes an RPC in XML [W3C.REC-xml-20001006] and sends it to a
   server using a secure, connection-oriented session. The server
   responds with a reply encoded in XML. The contents of both the
   request and the response are fully described in XML DTDs or XML
   schemas, or both, allowing both parties to recognize the syntax
   constraints imposed on the exchange.

   A key aspect of NETCONF is that it allows the functionality of the
   management protocol to closely mirror the native functionality of the
   device. This reduces implementation costs and allows timely access
   to new features. In addition, applications can access both the
   syntactic and semantic content of the device's native user interface.

   NETCONF allows a client to discover the set of protocol extensions
   supported by a server. These "capabilities" permit the client to
   adjust its behavior to take advantage of the features exposed by the
   device. The capability definitions can be easily extended in a
   noncentralized manner. Standard and non-standard capabilities can be
   defined with semantic and syntactic rigor. Capabilities are
   discussed in Section 8.

   The NETCONF protocol is a building block in a system of automated
   configuration. XML is the lingua franca of interchange, providing a
   flexible but fully specified encoding mechanism for hierarchical
   content. NETCONF can be used in concert with XML-based
   transformation technologies, such as XSLT [W3C.REC-xslt-19991116], to
   provide a system for automated generation of full and partial
   configurations. The system can query one or more databases for data
   about networking topologies, links, policies, customers, and
   services. This data can be transformed using one or more XSLT
   scripts from a task-oriented, vendor-independent data schema into a
   form that is specific to the vendor, product, operating system, and
   software release. The resulting data can be passed to the device
   using the NETCONF protocol.

   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].

1.1. Terminology

    o candidate configuration datastore: A configuration datastore that
       can be manipulated without impacting the device's current
       configuration and that can be committed to the running
       configuration datastore. Not all devices support a candidate
       configuration datastore.

    o capability: A functionality that supplements the base NETCONF
       specification.

    o client: A client invokes protocol operations on a server. In
       addition, a client can subscribe to receive notifications from a
       server.

    o configuration data: Configuration data is the set of writable data
       that is required to transform a system from its initial default
       state into its current state.

    o datastore: A conceptual place to store and access information. A
       datastore might be implemented, for example, using files, a
       database, flash memory locations or combinations thereof.

    o configuration datastore: A configuration datastore is defined as
       the datastore holding the complete set of configuration data that
       is required to get a device from its initial default state into a
       desired operational state.

    o message: A protocol element sent over a session. Messages are
       well-formed XML documents.

    o notification: A server initiated message indicating that a certain
       event has been recognized by the server.

    o protocol operation: A specific remote procedure call, as used
       within the NETCONF protocol.

    o remote procedure call: A remote procedure call (RPC), realized by
       exchanging <rpc> and <rpc-reply> messages.

    o running configuration datastore: A configuration datastore holding
       the complete configuration currently active on the device. The
       running configuration datastore always exists.

    o server: A server executes protocol operations invoked by a client.
       In addition, a server can send notifications to a client.

o   session: Client and server exchange messages using a secure,
    connection-oriented session.

o   startup configuration datastore: The configuration datastore
    holding the configuration loaded by the device when it boots.
    Only present on devices that separate the startup configuration
    datastore from the running configuration datastore.

o   state data: State data is the additional data on a system that is
    not configuration data such as read-only status information and
    collected statistics.

o   user: The authenticated identity of the client. The authenticated
    identity of a client is commonly referred to as the NETCONF
    username.

1.2. Protocol Overview

   NETCONF uses a simple RPC-based mechanism to facilitate communication
   between a client and a server. The client can be a script or
   application typically running as part of a network manager. The
   server is typically a network device. The terms "device" and
   "server" are used interchangeably in this document, as are "client"
   and "application".

   A NETCONF session is the logical connection between a network
   administrator or network configuration application and a network
   device. A device MUST support at least one NETCONF session and
   SHOULD support multiple sessions. Global configuration attributes
   can be changed during any authorized session, and the effects are
   visible in all sessions. Session-specific attributes affect only the
   session in which they are changed.

   NETCONF can be conceptually partitioned into four layers as shown in
   Figure 1.

```
           Layer                      Example
           +-------------+     +----------------+     +----------------+
    (4)    |   Content   |     | Configuration  |     |  Notification  |
           |             |     |      data      |     |      data      |
           +-------------+     +----------------+     +----------------+
                  |                    |                     |
           +-------------+     +----------------+            |
    (3)    |  Operations |     |  <edit-config> |            |
           |             |     |                |            |
           +-------------+     +----------------+            |
                  |                    |                     |
           +-------------+     +----------------+     +----------------+
    (2)    |   Messages  |     |     <rpc>,     |     | <notification> |
           |             |     |   <rpc-reply>  |     |                |
           +-------------+     +----------------+     +----------------+
                  |                    |                     |
           +-------------+     +--------------------------------------+
    (1)    |   Secure    |     | SSH, TLS, BEEP/TLS, SOAP/HTTP/TLS, ... |
           | Transports  |     |                                      |
           +-------------+     +--------------------------------------+
```
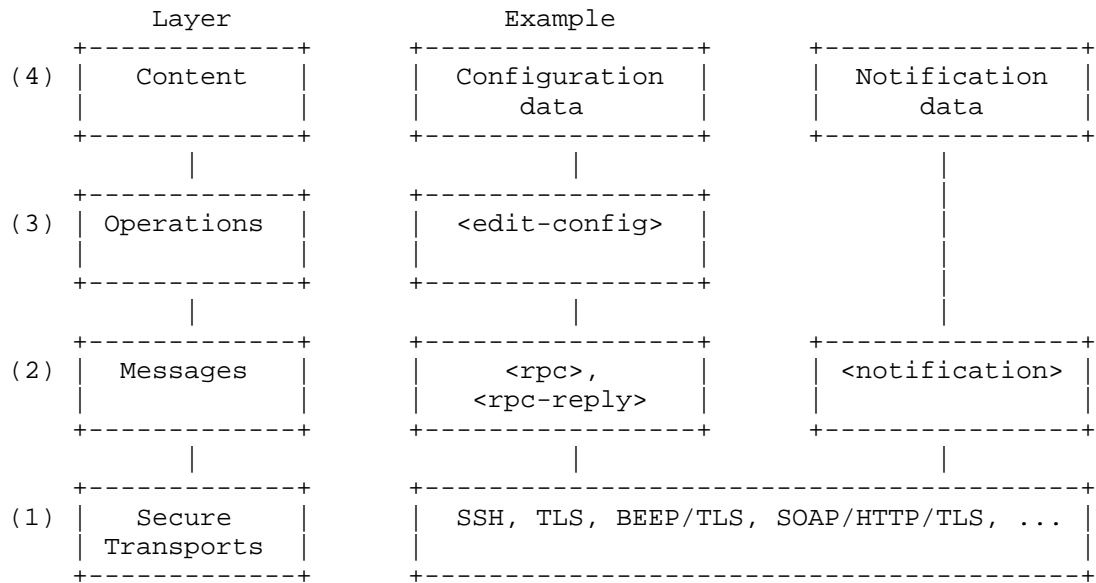
Figure 1: NETCONF Protocol Layers

1. The Secure Transport layer provides a communication path between
    the client and server. NETCONF can be layered over any transport
    protocol that provides a set of basic requirements. Section 2
    discusses these requirements.

2. The Messages layer provides a simple, transport-independent
    framing mechanism for encoding RPCs and notifications. Section 4
    documents the RPC messages, and [RFC5717] documents
    notifications.

3. The Operations layer defines a set of base protocol operations
    invoked as RPC methods with XML-encoded parameters. Section 7
    details the list of base protocol operations.

4. The Content layer is outside the scope of this document. It is
    expected that separate efforts to standardize NETCONF data models
    will be undertaken.

The YANG data modeling language [RFC6020] has been developed for
specifying NETCONF data models and protocol operations, covering the
Operations and the Content layers of Figure 1.

1.3. Capabilities

   A NETCONF capability is a set of functionality that supplements the
   base NETCONF specification. The capability is identified by a
   uniform resource identifier (URI) [RFC3986].

   Capabilities augment the base operations of the device, describing
   both additional operations and the content allowed inside operations.
   The client can discover the server's capabilities and use any
   additional operations, parameters, and content defined by those
   capabilities.

   The capability definition might name one or more dependent
   capabilities. To support a capability, the server MUST support any
   capabilities upon which it depends.

   Section 8 defines the capabilities exchange that allows the client to
   discover the server's capabilities. Section 8 also lists the set of
   capabilities defined in this document.

   Additional capabilities can be defined at any time in external
   documents, allowing the set of capabilities to expand over time.
   Standards bodies can define standardized capabilities, and
   implementations can define proprietary ones. A capability URI MUST
   sufficiently distinguish the naming authority to avoid naming
   collisions.

1.4. Separation of Configuration and State Data

   The information that can be retrieved from a running system is
   separated into two classes, configuration data and state data.
   Configuration data is the set of writable data that is required to
   transform a system from its initial default state into its current
   state. State data is the additional data on a system that is not
   configuration data such as read-only status information and collected
   statistics. When a device is performing configuration operations, a
   number of problems would arise if state data were included:

   o Comparisons of configuration data sets would be dominated by
      irrelevant entries such as different statistics.

   o Incoming data could contain nonsensical requests, such as attempts
      to write read-only data.

   o The data sets would be large.

   o Archived data could contain values for read-only data items,
      complicating the processing required to restore archived data.

To account for these issues, the NETCONF protocol recognizes the
difference between configuration data and state data and provides
operations for each. The <get-config> operation retrieves
configuration data only, while the <get> operation retrieves
configuration and state data.

Note that the NETCONF protocol is focused on the information required
to get the device into its desired running state. The inclusion of
other important, persistent data is implementation specific. For
example, user files and databases are not treated as configuration
data by the NETCONF protocol.

For example, if a local database of user authentication data is
stored on the device, it is an implementation-dependent matter
whether it is included in configuration data.

2. Transport Protocol Requirements

   NETCONF uses an RPC-based communication paradigm. A client sends a
   series of one or more RPC request messages, which cause the server to
   respond with a corresponding series of RPC reply messages.

   The NETCONF protocol can be layered on any transport protocol that
   provides the required set of functionality. It is not bound to any
   particular transport protocol, but allows a mapping to define how it
   can be implemented over any specific protocol.

   The transport protocol MUST provide a mechanism to indicate the
   session type (client or server) to the NETCONF protocol layer.

   This section details the characteristics that NETCONF requires from
   the underlying transport protocol.

2.1. Connection-Oriented Operation

   NETCONF is connection-oriented, requiring a persistent connection
   between peers. This connection MUST provide reliable, sequenced data
   delivery. NETCONF connections are long-lived, persisting between
   protocol operations.

   In addition, resources requested from the server for a particular
   connection MUST be automatically released when the connection closes,
   making failure recovery simpler and more robust. For example, when a
   lock is acquired by a client, the lock persists until either it is
   explicitly released or the server determines that the connection has
   been terminated. If a connection is terminated while the client
   holds a lock, the server can perform any appropriate recovery. The
   <lock> operation is further discussed in Section 7.5.

2.2. Authentication, Integrity, and Confidentiality

   NETCONF connections MUST provide authentication, data integrity,
   confidentiality, and replay protection. NETCONF depends on the
   transport protocol for this capability. A NETCONF peer assumes that
   appropriate levels of security and confidentiality are provided
   independently of this document. For example, connections could be
   encrypted using TLS [RFC5246] or SSH [RFC4251], depending on the
   underlying protocol.

   NETCONF connections MUST be authenticated. The transport protocol is
   responsible for authentication of the server to the client and
   authentication of the client to the server. A NETCONF peer assumes
   that the connection's authentication information has been validated
   by the underlying transport protocol using sufficiently trustworthy

mechanisms and that the peer's identity has been sufficiently proven.

One goal of NETCONF is to provide a programmatic interface to the device that closely follows the functionality of the device's native interface. Therefore, it is expected that the underlying protocol uses existing authentication mechanisms available on the device. For example, a NETCONF server on a device that supports RADIUS [RFC2865] might allow the use of RADIUS to authenticate NETCONF sessions.

The authentication process MUST result in an authenticated client identity whose permissions are known to the server. The authenticated identity of a client is commonly referred to as the NETCONF username. The username is a string of characters that match the "Char" production from section 2.2 of [W3C.REC-xml-20001006] . The algorithm used to derive the username is transport protocol specific and in addition specific to the authentication mechanism used by the transport protocol. The transport protocol MUST provide a username to be used by the other NETCONF layers.

The access permissions of a given client, identified by its NETCONF username, are part of the configuration of the NETCONF server. These permissions MUST be enforced during the remainder of the NETCONF session. The details how access control is configured is outside the scope of this document.

2.3. Mandatory Transport Protocol

A NETCONF implementation MUST support the SSH transport protocol mapping [I-D.ietf-netconf-rfc4742bis].

3. XML Considerations

   XML serves as the encoding format for NETCONF, allowing complex
   hierarchical data to be expressed in a text format that can be read,
   saved, and manipulated with both traditional text tools and tools
   specific to XML.

   All NETCONF messages MUST be well-formed XML, encoded in UTF-8
   [RFC3629]. If a peer receives an <rpc> message that is not well-
   formed XML or not encoded in UTF-8, it SHOULD reply with a
   "malformed-message" error. If a reply cannot be sent for any reason,
   the server MUST terminate the session.

   A NETCONF message MAY begin with an XML declaration (see section 2.8
   of [W3C.REC-xml-20001006]).

   This section discusses a small number of XML-related considerations
   pertaining to NETCONF.

3.1. Namespace

   All NETCONF protocol elements are defined in the following namespace:

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

   NETCONF capability names MUST be URIs [RFC3986]. NETCONF
   capabilities are discussed in Section 8.

3.2. Document Type Declarations

   Document type declarations (see section 2.8 of
   [W3C.REC-xml-20001006]) MUST NOT appear in NETCONF content.

4. RPC Model

   The NETCONF protocol uses an RPC-based communication model. NETCONF
   peers use <rpc> and <rpc-reply> elements to provide transport
   protocol-independent framing of NETCONF requests and responses.

   The syntax and XML encoding of the Messages layer RPCs are formally
   defined in the XML schema in Appendix B.

4.1. <rpc> Element

   The <rpc> element is used to enclose a NETCONF request sent from the
   client to the server.

   The <rpc> element has a mandatory attribute "message-id", which is a
   string chosen by the sender of the RPC that will commonly encode a
   monotonically increasing integer. The receiver of the RPC does not
   decode or interpret this string but simply saves it to be used as a
   "message-id" attribute in any resulting <rpc-reply> message. The
   sender MUST ensure that the "message-id" value is normalized
   according to the XML attribute value normalization rules defined in
   [W3C.REC-xml-20001006] if the sender wants the string to be returned
   unmodified. For example:

```
    <rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <some-method>
        <!-- method parameters here... -->
      </some-method>
    </rpc>
```

   If additional attributes are present in an <rpc> element, a NETCONF
   peer MUST return them unmodified in the <rpc-reply> element. This
   includes any "xmlns" attributes.

   The name and parameters of an RPC are encoded as the contents of the
   <rpc> element. The name of the RPC is an element directly inside the
   <rpc> element, and any parameters are encoded inside this element.

   The following example invokes a method called <my-own-method>, which
   has two parameters, <my-first-parameter>, with a value of "14", and
   <another-parameter>, with a value of "fred":

```
  <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <my-own-method xmlns="http://example.net/me/my-own/1.0">
      <my-first-parameter>14</my-first-parameter>
      <another-parameter>fred</another-parameter>
    </my-own-method>
  </rpc>
```

The following example invokes a <rock-the-house> method with a
<zip-code> parameter of "27606-0100":

```
  <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rock-the-house xmlns="http://example.net/rock/1.0">
      <zip-code>27606-0100</zip-code>
    </rock-the-house>
  </rpc>
```

The following example invokes the NETCONF <get> method with no
parameters:

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

4.2. <rpc-reply> Element

The <rpc-reply> message is sent in response to an <rpc> message.

The <rpc-reply> element has a mandatory attribute "message-id", which
is equal to the "message-id" attribute of the <rpc> for which this is
a response.

A NETCONF server MUST also return any additional attributes included
in the <rpc> element unmodified in the <rpc-reply> element.

The response data is encoded as one or more child elements to the
<rpc-reply> element.

For example:

The following <rpc> element invokes the NETCONF <get> method and
includes an additional attribute called "user-id". Note that the
"user-id" attribute is not in the NETCONF namespace. The returned
<rpc-reply> element returns the "user-id" attribute, as well as the
requested content.

```
    <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns:ex="http://example.net/content/1.0"
        ex:user-id="fred">
      <get/>
    </rpc>


    <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns:ex="http://example.net/content/1.0"
        ex:user-id="fred">
      <data>
        <!-- contents here... -->
      </data>
    </rpc-reply>
```

4.3. <rpc-error> Element

The <rpc-error> element is sent in <rpc-reply> messages if an error
occurs during the processing of an <rpc> request.

If a server encounters multiple errors during the processing of an
<rpc> request, the <rpc-reply> MAY contain multiple <rpc-error>
elements. However, a server is not required to detect or report more
than one <rpc-error> element, if a request contains multiple errors.
A server is not required to check for particular error conditions in
a specific sequence. A server MUST return an <rpc-error> element if
any error conditions occur during processing.

A server MUST NOT return application-level- or data-model-specific
error information in an <rpc-error> element for which the client does
not have sufficient access rights.

The <rpc-error> element includes the following information:

error-type: Defines the conceptual layer that the error occurred.
   Enumeration. One of:

   * transport (layer: Secure Transport)

   * rpc (layer: Messages)

   * protocol (layer: Operations)

   * application (layer: Content)

error-tag: Contains a string identifying the error condition. See
   Appendix A for allowed values.

error-severity: Contains a string identifying the error severity, as
   determined by the device. One of:

   * error

   * warning

   Note that there are no <error-tag> values defined in this document
   which utilize the "warning" enumeration. This is reserved for
   future use.

error-app-tag: Contains a string identifying the data-model-specific
   or implementation-specific error condition, if one exists. This
   element will not be present if no appropriate application error
   tag can be associated with a particular error condition. If a
   data-model specific and a implementation-specific error-app-tag
   both exist, then the data-model specific value MUST be used by the
   server.

error-path: Contains the absolute XPath [W3C.REC-xpath-19991116]
   expression identifying the element path to the node that is
   associated with the error being reported in a particular
   <rpc-error> element. This element will not be present if no
   appropriate payload element or datastore node can be associated
   with a particular error condition.

   The XPath expression is interpreted in the following context:

   * The set of namespace declarations are those in scope on the
      <rpc-error> element.

   * The set of variable bindings is empty.

   * The function library is the core function library.

   The context node depends on the node associated with the error
   being reported:

   * If a payload element can be associated with the error, the
      context node is the rpc request's document node (i.e., the
      <rpc> element).

   * Otherwise, the context node is the root of all data models,
      i.e., the node which has the top-level nodes from all data
      models as children.

   error-message: Contains a string suitable for human display that
      describes the error condition. This element will not be present
      if no appropriate message is provided for a particular error
      condition. This element SHOULD include an "xml:lang" attribute as
      defined in [W3C.REC-xml-20001006] and discussed in [RFC3470].

   error-info: Contains protocol- or data-model-specific error content.
      This element will not be present if no such error content is
      provided for a particular error condition. The list in Appendix A
      defines any mandatory error-info content for each error. After
      any protocol-mandated content, a data model definition MAY mandate
      that certain application-layer error information be included in
      the error-info container. An implementation MAY include
      additional elements to provide extended and/or implementation-
      specific debugging information.

   Appendix A enumerates the standard NETCONF errors.

   Example:

      An error is returned if an <rpc> element is received without a
      "message-id" attribute. Note that only in this case is it
      acceptable for the NETCONF peer to omit the "message-id" attribute
      in the <rpc-reply> element.

    <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get-config>
        <source>
          <running/>
        </source>
      </get-config>
    </rpc>

    <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <rpc-error>
        <error-type>rpc</error-type>
        <error-tag>missing-attribute</error-tag>
        <error-severity>error</error-severity>
        <error-info>
          <bad-attribute>message-id</bad-attribute>
          <bad-element>rpc</bad-element>
        </error-info>
      </rpc-error>
    </rpc-reply>

   The following <rpc-reply> illustrates the case of returning multiple

   <rpc-error> elements.

   Note that the data models used in the examples in this section use
   the <name> element to distinguish between multiple instances of the
   <interface> element.

```
     <rpc-reply message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
       <rpc-error>
         <error-type>application</error-type>
         <error-tag>invalid-value</error-tag>
         <error-severity>error</error-severity>
         <error-path xmlns:t="http://example.com/schema/1.2/config">
           /t:top/t:interface[t:name="Ethernet0/0"]/t:mtu
         </error-path>
         <error-message xml:lang="en">
           MTU value 25000 is not within range 256..9192
         </error-message>
       </rpc-error>
       <rpc-error>
         <error-type>application</error-type>
         <error-tag>invalid-value</error-tag>
         <error-severity>error</error-severity>
         <error-path xmlns:t="http://example.com/schema/1.2/config">
           /t:top/t:interface[t:name="Ethernet1/0"]/t:address/t:name
         </error-path>
         <error-message xml:lang="en">
           Invalid IP address for interface Ethernet1/0
         </error-message>
       </rpc-error>
     </rpc-reply>
```


4.4. <ok> Element

   The <ok> element is sent in <rpc-reply> messages if no errors or
   warnings occurred during the processing of an <rpc> request, and no
   data was returned from the operation. For example:

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

4.5. Pipelining

   NETCONF <rpc> requests MUST be processed serially by the managed
   device. Additional <rpc> requests MAY be sent before previous ones
   have been completed. The managed device MUST send responses only in
   the order the requests were received.

5. Configuration Model

   NETCONF provides an initial set of operations and a number of
   capabilities that can be used to extend the base. NETCONF peers
   exchange device capabilities when the session is initiated as
   described in Section 8.1.

5.1. Configuration Datastores

   NETCONF defines the existence of one or more configuration datastores
   and allows configuration operations on them. A configuration
   datastore is defined as the complete set of configuration data that
   is required to get a device from its initial default state into a
   desired operational state. The configuration datastore does not
   include state data or executive commands.

   The running configuration datastore holds the complete configuration
   currently active on the network device. Only one configuration
   datastore of this type exists on the device, and it is always
   present. NETCONF protocol operations refer to this datastore using
   the <running> element.

   Only the <running> configuration datastore is present in the base
   model. Additional configuration datastores MAY be defined by
   capabilities. Such configuration datastores are available only on
   devices that advertise the capabilities.

   The capabilities in Sections 8.3 and 8.7 define the <candidate> and
   <startup> configuration datastores, respectively.

5.2. Data Modeling

   Data modeling and content issues are outside the scope of the NETCONF
   protocol. An assumption is made that the device's data model is
   well-known to the application and that both parties are aware of
   issues such as the layout, containment, keying, lookup, replacement,
   and management of the data, as well as any other constraints imposed
   by the data model.

   NETCONF carries configuration data inside the <config> element that
   is specific to device's data model. The protocol treats the contents
   of that element as opaque data. The device uses capabilities to
   announce the set of data models that the device implements. The
   capability definition details the operation and constraints imposed
   by data model.

   Devices and managers can support multiple data models, including both
   standard and proprietary data models.

6. Subtree Filtering

6.1. Overview

   XML subtree filtering is a mechanism that allows an application to
   select particular XML subtrees to include in the <rpc-reply> for a
   <get> or <get-config> operation. A small set of filters for
   inclusion, simple content exact-match, and selection is provided,
   which allows some useful, but also very limited, selection
   mechanisms. The server does not need to utilize any data-model-
   specific semantics during processing, allowing for simple and
   centralized implementation strategies.

   Conceptually, a subtree filter is comprised of zero or more element
   subtrees, which represent the filter selection criteria. At each
   containment level within a subtree, the set of sibling nodes is
   logically processed by the server to determine if its subtree and
   path of elements to the root are included in the filter output.

   Each node specified in a subtree filter represents an inclusive
   filter. Only associated nodes in underlying data model(s) within the
   specified datastore on the server are selected by the filter. A node
   is selected if it matches the selection criteria and hierarchy of
   elements given in the filter data, except that the filter absolute
   path name is adjusted to start from the layer below <filter>.

   Response messages contain only the subtrees selected by the filter.
   Any selection criteria that were present in the request, within a
   particular selected subtree, are also included in the response. Note
   that some elements expressed in the filter as leaf nodes will be
   expanded (i.e., subtrees included) in the filter output. Specific
   data instances are not duplicated in the response in the event that
   the request contains multiple filter subtree expressions that select
   the same data.

6.2. Subtree Filter Components

   A subtree filter is comprised of XML elements and their XML
   attributes. There are five types of components that can be present
   in a subtree filter:

   o Namespace Selection

   o Attribute Match Expressions

   o Containment Nodes

o Selection Nodes

o Content Match Nodes

6.2.1. Namespace Selection

   A namespace is considered to match (for filter purposes) if the XML
   namespace associated with a particular node within the <filter>
   element is the same as in the underlying data model. Note that
   namespace selection cannot be used by itself. At least one element
   MUST be specified in the filter if any elements are to be included in
   the filter output.

   An XML namespace wildcard mechanism is defined for subtree filtering.
   If an element within the <filter> element is not qualified by a
   namespace (e.g., xmlns=""), then the server MUST evaluate all the XML
   namespaces it supports, when processing that subtree filter node.
   This wildcard mechanism is not applicable to XML attributes.

   Note that prefix values for qualified namespaces are not relevant
   when comparing filter elements to elements in the underlying data
   model.

   Example:

```
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config"/>
</filter>
```

   In this example, the <top> element is a selection node, and only this
   node in the "http://example.com/schema/1.2/config" namespace and any
   child nodes (from the underlying data model) will be included in the
   filter output.

6.2.2. Attribute Match Expressions

   An attribute that appears in a subtree filter is part of an
   "attribute match expression". Any number of (unqualified or
   qualified) XML attributes MAY be present in any type of filter node.
   In addition to the selection criteria normally applicable to that
   node, the selected data MUST have matching values for every attribute
   specified in the node. If an element is not defined to include a
   specified attribute, then it is not selected in the filter output.

   Example:

```
  <filter type="subtree">
    <t:top xmlns:t="http://example.com/schema/1.2/config">
      <t:interfaces>
        <t:interface t:ifName="eth0"/>
      </t:interfaces>
    </t:top>
  </filter>
```

In this example, the <top>, and <interfaces> elements are containment nodes, the <interface> element is a selection node, and "ifName" is an attribute match expression. Only "interface" nodes in the "http://example.com/schema/1.2/config" namespace that have an "ifName" attribute with the value "eth0" and occur within "interfaces" nodes within "top" nodes will be included in the filter output.

6.2.3. Containment Nodes

Nodes that contain child elements within a subtree filter are called "containment nodes". Each child element can be any type of node, including another containment node. For each containment node specified in a subtree filter, all data model instances that exactly match the specified namespaces, element hierarchy, and any attribute match expressions are included in the filter output.

Example:

```
  <filter type="subtree">
    <top xmlns="http://example.com/schema/1.2/config">
      <users/>
    </top>
  </filter>
```

In this example, the <top> element is a containment node.

6.2.4. Selection Nodes

An empty leaf node within a filter is called a "selection node", and it represents an "explicit selection" filter on the underlying data model. Presence of any selection nodes within a set of sibling nodes will cause the filter to select the specified subtree(s) and suppress automatic selection of the entire set of sibling nodes in the underlying data model. For filtering purposes, an empty leaf node can be declared either with an empty tag (e.g., <foo/>) or with explicit start and end tags (e.g., <foo> </foo>). Any whitespace characters are ignored in this form.

Example:

```
   <filter type="subtree">
     <top xmlns="http://example.com/schema/1.2/config">
       <users/>
     </top>
   </filter>
```

In this example, the <top> element is a containment node, and the
<users> element is a selection node. Only "users" nodes in the
"http://example.com/schema/1.2/config" namespace that occur within a
<top> element that is the root of the configuration datastore will be
included in the filter output.

6.2.5. Content Match Nodes

A leaf node that contains simple content is called a "content match
node". It is used to select some or all of its sibling nodes for
filter output, and it represents an exact-match filter on the leaf
node element content. The following constraints apply to content
match nodes:

o A content match node MUST NOT contain nested elements.

o Multiple content match nodes (i.e., sibling nodes) are logically
   combined in an "AND" expression.

o Filtering of mixed content is not supported.

o Filtering of list content is not supported.

o Filtering of whitespace-only content is not supported.

o A content match node MUST contain non-whitespace characters. An
   empty element (e.g., <foo></foo>) will be interpreted as a
   selection node (e.g., <foo/>).

o Leading and trailing whitespace characters are ignored, but any
   whitespace characters within a block of text characters are not
   ignored or modified.

If all specified sibling content match nodes in a subtree filter
expression are "true", then the filter output nodes are selected in
the following manner:

o Each content match node in the sibling set is included in the
   filter output.

o If any containment nodes are present in the sibling set, then they
   are processed further and included if any nested filter criteria

are also met.

o If any selection nodes are present in the sibling set, then all of
   them are included in the filter output.

o If any sibling nodes of the selection node are instance identifier
   components for a conceptual data structure (e.g., list key leaf),
   then they MAY also be included in the filter output.

o Otherwise (i.e., there are no selection or containment nodes in
   the filter sibling set), all the nodes defined at this level in
   the underlying data model (and their subtrees, if any) are
   returned in the filter output.

If any of the sibling content match node tests are "false", then no
further filter processing is performed on that sibling set, and none
of the sibling subtrees are selected by the filter, including the
content match node(s).

Example:

```
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config">
    <users>
      <user>
        <name>fred</name>
      </user>
    </users>
  </top>
</filter>
```

In this example, the <users> and <user> nodes are both containment
nodes, and <name> is a content match node. Since no sibling nodes of
<name> are specified (and therefore no containment or selection
nodes), all of the sibling nodes of <name> are returned in the filter
output. Only "user" nodes in the
"http://example.com/schema/1.2/config" namespace that match the
element hierarchy and for which the <name> element is equal to "fred"
will be included in the filter output.

6.3. Subtree Filter Processing

The filter output (the set of selected nodes) is initially empty.

Each subtree filter can contain one or more data model fragments,
which represent portions of the data model that will be selected
(with all child nodes) in the filter output.

   Each subtree data fragment is compared by the server to the internal
   data models supported by the server. If the entire subtree data-
   fragment filter (starting from the root to the innermost element
   specified in the filter) exactly matches a corresponding portion of
   the supported data model, then that node and all its children are
   included in the result data.

   The server processes all nodes with the same parent node (sibling
   set) together, starting from the root to the leaf nodes. The root
   elements in the filter are considered in the same sibling set
   (assuming they are in the same namespace), even though they do not
   have a common parent.

   For each sibling set, the server determines which nodes are included
   (or potentially included) in the filter output, and which sibling
   subtrees are excluded (pruned) from the filter output. The server
   first determines which types of nodes are present in the sibling set
   and processes the nodes according to the rules for their type. If
   any nodes in the sibling set are selected, then the process is
   recursively applied to the sibling sets of each selected node. The
   algorithm continues until all sibling sets in all subtrees specified
   in the filter have been processed.

6.4. Subtree Filtering Examples

6.4.1. No Filter

   Leaving out the filter on the <get> operation returns the entire data
   model.

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

<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <!-- ... entire set of data returned ... -->
  </data>
</rpc-reply>
```

6.4.2. Empty Filter

   An empty filter will select nothing because no content match or
   selection nodes are present. This is not an error. The <filter>
   element's "type" attribute used in these examples is discussed
   further in Section 7.1.

```
      <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get>
          <filter type="subtree">
          </filter>
        </get>
      </rpc>

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

6.4.3. Select the Entire <users> Subtree

   The filter in this example contains one selection node (<users>), so
   just that subtree is selected by the filter. This example represents
   the fully-populated <users> data model in most of the filter examples
   that follow. In a real data model, the <company-info> would not
   likely be returned with the list of users for a particular host or
   network.

   NOTE: The filtering and configuration examples used in this document
   appear in the namespace "http://example.com/schema/1.2/config". The
   root element of this namespace is <top>. The <top> element and its
   descendents represent an example configuration data model only.

```
      <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get-config>
          <source>
            <running/>
          </source>
          <filter type="subtree">
            <top xmlns="http://example.com/schema/1.2/config">
              <users/>
            </top>
          </filter>
        </get-config>
      </rpc>


      <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <data>
          <top xmlns="http://example.com/schema/1.2/config">
            <users>
```

```
            <user>
              <name>root</name>
              <type>superuser</type>
              <full-name>Charlie Root</full-name>
              <company-info>
                <dept>1</dept>
                <id>1</id>
              </company-info>
            </user>
            <user>
              <name>fred</name>
              <type>admin</type>
              <full-name>Fred Flintstone</full-name>
              <company-info>
                <dept>2</dept>
                <id>2</id>
              </company-info>
            </user>
            <user>
              <name>barney</name>
              <type>admin</type>
              <full-name>Barney Rubble</full-name>
              <company-info>
                <dept>2</dept>
                <id>3</id>
              </company-info>
            </user>
          </users>
        </top>
      </data>
    </rpc-reply>
```

   The following filter request would have produced the same result, but
   only because the container <users> defines one child element
   (<user>).

```
  <rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
      <source>
        <running/>
      </source>
      <filter type="subtree">
        <top xmlns="http://example.com/schema/1.2/config">
          <users>
            <user/>
          </users>
        </top>
      </filter>
    </get-config>
  </rpc>
```

6.4.4. Select All <name> Elements within the <users> Subtree

   This filter contains two containment nodes (<users>, <user>) and one
   selection node (<name>). All instances of the <name> element in the
   same sibling set are selected in the filter output. The client might
   need to know that <name> is used as an instance identifier in this
   particular data structure, but the server does not need to know that
   meta-data in order to process the request.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user>
            <name/>
          </user>
        </users>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://example.com/schema/1.2/config">
      <users>
        <user>
          <name>root</name>
        </user>
        <user>
          <name>fred</name>
        </user>
        <user>
          <name>barney</name>
        </user>
      </users>
    </top>
  </data>
</rpc-reply>
```

6.4.5. One Specific <user> Entry

   This filter contains two containment nodes (<users>, <user>) and one
   content match node (<name>). All instances of the sibling set
   containing <name> for which the value of <name> equals "fred" are
   selected in the filter output.

```
   <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get-config>
       <source>
         <running/>
       </source>
       <filter type="subtree">
         <top xmlns="http://example.com/schema/1.2/config">
           <users>
             <user>
               <name>fred</name>
             </user>
           </users>
         </top>
       </filter>
     </get-config>
   </rpc>

   <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <data>
       <top xmlns="http://example.com/schema/1.2/config">
         <users>
           <user>
             <name>fred</name>
             <type>admin</type>
             <full-name>Fred Flintstone</full-name>
             <company-info>
               <dept>2</dept>
               <id>2</id>
             </company-info>
           </user>
         </users>
       </top>
     </data>
   </rpc-reply>
```

6.4.6. Specific Elements from a Specific <user> Entry

   This filter contains two containment nodes (<users>, <user>), one
   content match node (<name>), and two selection nodes (<type>,
   <full-name>). All instances of the <type> and <full-name> elements
   in the same sibling set containing <name> for which the value of
   <name> equals "fred" are selected in the filter output. The
   <company-info> element is not included because the sibling set
   contains selection nodes.

```
      <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get-config>
          <source>
            <running/>
          </source>
          <filter type="subtree">
            <top xmlns="http://example.com/schema/1.2/config">
              <users>
                <user>
                  <name>fred</name>
                  <type/>
                  <full-name/>
                </user>
              </users>
            </top>
          </filter>
        </get-config>
      </rpc>

      <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <data>
          <top xmlns="http://example.com/schema/1.2/config">
            <users>
              <user>
                <name>fred</name>
                <type>admin</type>
                <full-name>Fred Flintstone</full-name>
              </user>
            </users>
          </top>
        </data>
      </rpc-reply>
```

6.4.7. Multiple Subtrees

   This filter contains three subtrees (name=root, fred, barney).

   The "root" subtree filter contains two containment nodes (<users>,
   <user>), one content match node (<name>), and one selection node
   (<company-info>). The subtree selection criteria is met, and just
   the company-info subtree for "root" is selected in the filter output.

   The "fred" subtree filter contains three containment nodes (<users>,
   <user>, <company-info>), one content match node (<name>), and one
   selection node (<id>). The subtree selection criteria is met, and
   just the <id> element within the company-info subtree for "fred" is

selected in the filter output.

The "barney" subtree filter contains three containment nodes
(<users>, <user>, <company-info>), two content match nodes (<name>,
<type>), and one selection node (<dept>). The subtree selection
criteria is not met because user "barney" is not a "superuser", and
the entire subtree for "barney" (including its parent <user> entry)
is excluded from the filter output.

```
  <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
      <source>
        <running/>
      </source>
      <filter type="subtree">
        <top xmlns="http://example.com/schema/1.2/config">
          <users>
            <user>
              <name>root</name>
              <company-info/>
            </user>
            <user>
              <name>fred</name>
              <company-info>
                <id/>
              </company-info>
            </user>
            <user>
              <name>barney</name>
              <type>superuser</type>
              <company-info>
                <dept/>
              </company-info>
            </user>
          </users>
        </top>
      </filter>
    </get-config>
  </rpc>

  <rpc-reply message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user>
            <name>root</name>
```

```
                <company-info>
                  <dept>1</dept>
                  <id>1</id>
                </company-info>
              </user>
              <user>
                <name>fred</name>
                <company-info>
                  <id>2</id>
                </company-info>
              </user>
            </users>
          </top>
        </data>
      </rpc-reply>
```

6.4.8. Elements with Attribute Naming

   In this example, the filter contains one containment node
   (<interfaces>), one attribute match expression ("ifName"), and one
   selection node (<interface>). All instances of the <interface>
   subtree that have an "ifName" attribute equal to "eth0" are selected
   in the filter output. The filter data elements and attributes are
   qualified because the "ifName" attribute will not be considered part
   of the "schema/1.2" namespace if it is unqualified.

```
      <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get>
          <filter type="subtree">
            <t:top xmlns:t="http://example.com/schema/1.2/stats">
              <t:interfaces>
                <t:interface t:ifName="eth0"/>
              </t:interfaces>
            </t:top>
          </filter>
        </get>
      </rpc>

      <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <data>
          <t:top xmlns:t="http://example.com/schema/1.2/stats">
            <t:interfaces>
              <t:interface t:ifName="eth0">
                <t:ifInOctets>45621</t:ifInOctets>
                <t:ifOutOctets>774344</t:ifOutOctets>
              </t:interface>
            </t:interfaces>
          </t:top>
        </data>
      </rpc-reply>
```

   If "ifName" were a child node instead of an attribute, then the
   following request would produce similar results.

```
      <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get>
          <filter type="subtree">
            <top xmlns="http://example.com/schema/1.2/stats">
              <interfaces>
                <interface>
                  <ifName>eth0</ifName>
                </interface>
              </interfaces>
            </top>
          </filter>
        </get>
      </rpc>
```

7. Protocol Operations

   The NETCONF protocol provides a small set of low-level operations to
   manage device configurations and retrieve device state information.
   The base protocol provides operations to retrieve, configure, copy,
   and delete configuration datastores. Additional operations are
   provided, based on the capabilities advertised by the device.

   The base protocol includes the following protocol operations:

   o get

   o get-config

   o edit-config

   o copy-config

   o delete-config

   o lock

   o unlock

   o close-session

   o kill-session

   A protocol operation can fail for various reasons, including
   "operation not supported". An initiator SHOULD NOT assume that any
   operation will always succeed. The return values in any RPC reply
   SHOULD be checked for error responses.

   The syntax and XML encoding of the protocol operations are formally
   defined in the YANG module in Appendix C. The following sections
   describe the semantics of each protocol operation.

7.1. <get-config>

   Description:

      Retrieve all or part of a specified configuration datastore.

   Parameters:

   source:

      Name of the configuration datastore being queried, such as
      <running/>.

   filter:

      This parameter identifies the portions of the device
      configuration datastore to retrieve. If this parameter is not
      present, the entire configuration is returned.

      The <filter> element MAY optionally contain a "type" attribute.
      This attribute indicates the type of filtering syntax used
      within the <filter> element. The default filtering mechanism
      in NETCONF is referred to as subtree filtering and is described
      in Section 6. The value "subtree" explicitly identifies this
      type of filtering.

      If the NETCONF peer supports the :xpath capability
      (Section 8.9), the value "xpath" MAY be used to indicate that
      the "select" attribute on the <filter> element contains an
      XPath expression.

Positive Response:

   If the device can satisfy the request, the server sends an
   <rpc-reply> element containing a <data> element with the results
   of the query.

Negative Response:

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

Example: To retrieve the entire <users> subtree:

```
    <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get-config>
        <source>
          <running/>
        </source>
        <filter type="subtree">
          <top xmlns="http://example.com/schema/1.2/config">
            <users/>
          </top>
        </filter>
      </get-config>
    </rpc>

    <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
        <top xmlns="http://example.com/schema/1.2/config">
          <users>
            <user>
              <name>root</name>
              <type>superuser</type>
              <full-name>Charlie Root</full-name>
              <company-info>
                <dept>1</dept>
                <id>1</id>
              </company-info>
            </user>
            <!-- additional <user> elements appear here... -->
          </users>
        </top>
      </data>
    </rpc-reply>
```

   Section 6 contains additional examples of subtree filtering.

7.2. <edit-config>

   Description:

      The <edit-config> operation loads all or part of a specified
      configuration to the specified target configuration datastore.
      This operation allows the new configuration to be expressed in
      several ways, such as using a local file, a remote file, or
      inline. If the target configuration datastore does not exist, it
      will be created.

      If a NETCONF peer supports the :url capability (Section 8.8), the

<url> element can appear instead of the <config> parameter.

The device analyzes the source and target configurations and
performs the requested changes. The target configuration is not
necessarily replaced, as with the <copy-config> message. Instead,
the target configuration is changed in accordance with the
source's data and requested operations.

If the <edit-config> operation contains multiple sub-operations
which apply to the same conceptual node in the underlying data
model, then the result of the operation is undefined (i.e.,
outside the scope of the NETCONF protocol).

Attributes:

operation:

 Elements in the <config> subtree MAY contain an "operation"
 attribute. The attribute identifies the point in the
 configuration to perform the operation and MAY appear on
 multiple elements throughout the <config> subtree.

 If the "operation" attribute is not specified, the
 configuration is merged into the configuration datastore.

 The "operation" attribute has one of the following values:

 merge: The configuration data identified by the element
  containing this attribute is merged with the configuration
  at the corresponding level in the configuration datastore
  identified by the <target> parameter. This is the default
  behavior.

 replace: The configuration data identified by the element
  containing this attribute replaces any related configuration
  in the configuration datastore identified by the <target>
  parameter. If no such configuration data exists in the
  configuration datastore, it is created. Unlike a
  <copy-config> operation, which replaces the entire target
  configuration, only the configuration actually present in
  the <config> parameter is affected.

 create: The configuration data identified by the element
  containing this attribute is added to the configuration if
  and only if the configuration data does not already exist in
  the configuration datastore. If the configuration data
  exists, an <rpc-error> element is returned with an
  <error-tag> value of "data-exists".

delete: The configuration data identified by the element
    containing this attribute is deleted from the configuration
    if and only if the configuration data currently exists in
    the configuration datastore. If the configuration data does
    not exist, an <rpc-error> element is returned with an
    <error-tag> value of "data-missing".

remove: The configuration data identified by the element
    containing this attribute is deleted from the configuration
    if the configuration data currently exists in the
    configuration datastore. If the configuration data does not
    exist, the "remove" operation is silently ignored by the
    server.

Parameters:

target:

    Name of the configuration datastore being edited, such as
    <running/> or <candidate/>.

default-operation:

    Selects the default operation (as described in the "operation"
    attribute) for this <edit-config> request. The default value
    for the <default-operation> parameter is "merge".

    The <default-operation> parameter is optional, but if provided,
    it has one of the following values:

    merge: The configuration data in the <config> parameter is
        merged with the configuration at the corresponding level in
        the target datastore. This is the default behavior.

    replace: The configuration data in the <config> parameter
        completely replaces the configuration in the target
        datastore. This is useful for loading previously saved
        configuration data.

    none: The target datastore is unaffected by the configuration
        in the <config> parameter, unless and until the incoming
        configuration data uses the "operation" attribute to request
        a different operation. If the configuration in the <config>
        parameter contains data for which there is not a
        corresponding level in the target datastore, an <rpc-error>
        is returned with an <error-tag> value of data-missing.
        Using "none" allows operations like "delete" to avoid
        unintentionally creating the parent hierarchy of the element

      to be deleted.

   test-option:

      The <test-option> element MAY be specified only if the device
      advertises the :validate:1.1 capability (Section 8.6).

      The <test-option> element has one of the following values:

      test-then-set: Perform a validation test before attempting to
         set. If validation errors occur, do not perform the
         <edit-config> operation. This is the default test-option.

      set: Perform a set without a validation test first.

      test-only: Perform only the validation test, without
         attempting to set.

   error-option:

      The <error-option> element has one of the following values:

      stop-on-error: Abort the edit-config operation on first error.
         This is the default error-option.

      continue-on-error: Continue to process configuration data on
         error; error is recorded, and negative response is generated
         if any errors occur.

      rollback-on-error: If an error condition occurs such that an
         error severity <rpc-error> element is generated, the server
         will stop processing the edit-config operation and restore
         the specified configuration to its complete state at the
         start of this edit-config operation. This option requires
         the server to support the :rollback-on-error capability
         described in Section 8.5.

   config:

      A hierarchy of configuration data as defined by one of the
      device's data models. The contents MUST be placed in an
      appropriate namespace, to allow the device to detect the
      appropriate data model, and the contents MUST follow the
      constraints of that data model, as defined by its capability
      definition. Capabilities are discussed in Section 8.

   Positive Response:

      If the device was able to satisfy the request, an <rpc-reply> is
      sent containing an <ok> element.

   Negative Response:

      An <rpc-error> response is sent if the request cannot be completed
      for any reason.

   Example:

      The <edit-config> examples in this section utilize a simple data
      model, in which multiple instances of the <interface> element can
      be present, and an instance is distinguished by the <name> element
      within each <interface> element.

      Set the MTU to 1500 on an interface named "Ethernet0/0" in the
      running configuration:

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

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

   Add an interface named "Ethernet0/0" to the running configuration,
   replacing any previous interface with that name:

```
 <rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
     <target>
       <running/>
     </target>
     <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
       <top xmlns="http://example.com/schema/1.2/config">
         <interface xc:operation="replace">
           <name>Ethernet0/0</name>
           <mtu>1500</mtu>
           <address>
             <name>192.0.2.4</name>
             <prefix-length>24</prefix-length>
           </address>
         </interface>
       </top>
     </config>
   </edit-config>
 </rpc>

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

Delete the configuration for an interface named "Ethernet0/0" from
the running configuration:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>none</default-operation>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <top xmlns="http://example.com/schema/1.2/config">
        <interface xc:operation="delete">
          <name>Ethernet0/0</name>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

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

Delete interface 192.0.2.4 from an OSPF area (other interfaces
configured in the same area are unaffected):

```
    <rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <edit-config>
        <target>
          <running/>
        </target>
        <default-operation>none</default-operation>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <top xmlns="http://example.com/schema/1.2/config">
            <protocols>
              <ospf>
                <area>
                  <name>0.0.0.0</name>
                  <interfaces>
                    <interface xc:operation="delete">
                      <name>192.0.2.4</name>
                    </interface>
                  </interfaces>
                </area>
              </ospf>
            </protocols>
          </top>
        </config>
      </edit-config>
    </rpc>

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

7.3. <copy-config>

   Description:

      Create or replace an entire configuration datastore with the
      contents of another complete configuration datastore. If the
      target datastore exists, it is overwritten. Otherwise, a new one
      is created, if allowed.

      If a NETCONF peer supports the :url capability (Section 8.8), the
      <url> element can appear as the <source> or <target> parameter.

      Even if it advertises the :writable-running capability, a device
      MAY choose not to support the <running/> configuration datastore
      as the <target> parameter of a <copy-config> operation. A device
      MAY choose not to support remote-to-remote copy operations, where
      both the <source> and <target> parameters use the <url> element.

If the <source> and <target> parameters identify the same URL or
configuration datastore, an error MUST be returned with an error-
tag containing "invalid-value".

Parameters:

   target:

      Name of the configuration datastore to use as the destination
      of the <copy-config> operation.

   source:

      Name of the configuration datastore to use as the source of the
      <copy-config> operation, or the <config> element containing the
      complete configuration to copy.

Positive Response:

   If the device 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.

Example:

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <running/>
    </target>
    <source>
      <url>https://user:password@example.com/cfg/new.txt</url>
    </source>
  </copy-config>
</rpc>

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

7.4. <delete-config>

   Description:

      Delete a configuration datastore. The <running> configuration
      datastore cannot be deleted.

      If a NETCONF peer supports the :url capability (Section 8.8), the
      <url> element can appear as the <target> parameter.

   Parameters:

      target:

         Name of the configuration datastore to delete.

   Positive Response:

      If the device 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.

   Example:

     <rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <delete-config>
         <target>
           <startup/>
         </target>
       </delete-config>
     </rpc>

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

7.5. <lock>

Description:

The <lock> operation allows the client to lock the entire
configuration datastore system of a device. Such locks are
intended to be short-lived and allow a client to make a change
without fear of interaction with other NETCONF clients, non-
NETCONF clients (e.g., SNMP and command line interface (CLI)
scripts), and human users.

An attempt to lock the configuration datastore MUST fail if an
existing session or other entity holds a lock on any portion of
the lock target.

When the lock is acquired, the server MUST prevent any changes to
the locked resource other than those requested by this session.
SNMP and CLI requests to modify the resource MUST fail with an
appropriate error.

The duration of the lock is defined as beginning when the lock is
acquired and lasting until either the lock is released or the
NETCONF session closes. The session closure can be explicitly
performed by the client, or implicitly performed by the server
based on criteria such as failure of the underlying transport,
simple inactivity timeout, or detection of abusive behavior on the
part of the client . This criteria is dependent on the
implementation and the underlying transport.

The <lock> operation takes a mandatory parameter, <target>. The
<target> parameter names the configuration datastore that will be
locked. When a lock is active, using the <edit-config> operation
on the locked configuration datastore and using the locked
configuration as a target of the <copy-config> operation will be
disallowed by any other NETCONF session. Additionally, the system
will ensure that these locked configuration resources will not be
modified by other non-NETCONF management operations such as SNMP
and CLI. The <kill-session> operation can be used to force the
release of a lock owned by another NETCONF session. It is beyond
the scope of this document to define how to break locks held by
other entities.

A lock MUST NOT be granted if either of the following conditions
is true:

* A lock is already held by any NETCONF session or another
   entity.

* The target configuration is <candidate>, it has already been
   modified, and these changes have not been committed or rolled

      back.

   The server MUST respond with either an <ok> element or an
   <rpc-error>.

   A lock will be released by the system if the session holding the
   lock is terminated for any reason.

Parameters:

   target:

      Name of the configuration datastore to lock.

Positive Response:

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

Negative Response:

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

   If the lock is already held, the <error-tag> element will be
   "lock-denied" and the <error-info> element will include the
   <session-id> of the lock owner. If the lock is held by a non-
   NETCONF entity, a <session-id> of 0 (zero) is included. Note that
   any other entity performing a lock on even a partial piece of a
   target will prevent a NETCONF lock (which is global) from being
   obtained on that target.

Example:

   The following example shows a successful acquisition of a lock.

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

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

```
      </rpc-reply>


   Example:

      The following example shows a failed attempt to acquire a lock
      when the lock is already in use.


   <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <lock>
       <target>
         <running/>
       </target>
     </lock>
   </rpc>

   <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <rpc-error> <!-- lock failed -->
       <error-type>protocol</error-type>
       <error-tag>lock-denied</error-tag>
       <error-severity>error</error-severity>
       <error-message>
         Lock failed, lock is already held
       </error-message>
       <error-info>
         <session-id>454</session-id>
         <!-- lock is held by NETCONF session 454 -->
       </error-info>
     </rpc-error>
   </rpc-reply>
```

7.6. <unlock>

   Description:

      The <unlock> operation is used to release a configuration lock,
      previously obtained with the <lock> operation.

      An <unlock> operation will not succeed if any of the following
      conditions are true:

      * the specified lock is not currently active

      * the session issuing the <unlock> operation is not the same
         session that obtained the lock

The server MUST respond with either an <ok> element or an
<rpc-error>.

Parameters:

target:

Name of the configuration datastore to unlock.

A NETCONF client is not permitted to unlock a configuration
datastore that it did not lock.

Positive Response:

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

Negative Response:

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

Example:

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

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

7.7. <get>

Description:

Retrieve running configuration and device state information.

Parameters:

filter:

> This parameter specifies the portion of the system
> configuration and state data to retrieve. If this parameter is
> not present, all the device configuration and state information
> is returned.
>
> The <filter> element MAY optionally contain a "type" attribute.
> This attribute indicates the type of filtering syntax used
> within the <filter> element. The default filtering mechanism
> in NETCONF is referred to as subtree filtering and is described
> in Section 6. The value "subtree" explicitly identifies this
> type of filtering.
>
> If the NETCONF peer supports the :xpath capability
> (Section 8.9), the value "xpath" MAY be used to indicate that
> the "select" attribute of the <filter> element contains an
> XPath expression.

Positive Response:

> If the device was able to satisfy the request, an <rpc-reply> is
> sent. The <data> section contains the appropriate subset.

Negative Response:

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

Example:

```
    <rpc message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get>
        <filter type="subtree">
          <top xmlns="http://example.com/schema/1.2/stats">
            <interfaces>
              <interface>
                <ifName>eth0</ifName>
              </interface>
            </interfaces>
          </top>
        </filter>
      </get>
    </rpc>

    <rpc-reply message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
        <top xmlns="http://example.com/schema/1.2/stats">
          <interfaces>
            <interface>
              <ifName>eth0</ifName>
              <ifInOctets>45621</ifInOctets>
              <ifOutOctets>774344</ifOutOctets>
            </interface>
          </interfaces>
        </top>
      </data>
    </rpc-reply>
```

7.8. <close-session>

   Description:

      Request graceful termination of a NETCONF session.

      When a NETCONF server receives a <close-session> request, it will
      gracefully close the session. The server will release any locks
      and resources associated with the session and gracefully close any
      associated connections. Any NETCONF requests received after a
      <close-session> request will be ignored.

   Positive Response:

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

   Negative Response:

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

   Example:

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

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

7.9. <kill-session>

   Description:

      Force the termination of a NETCONF session.

      When a NETCONF entity receives a <kill-session> request for an
      open session, it will abort any operations currently in process,
      release any locks and resources associated with the session, and
      close any associated connections.

      If a NETCONF server receives a <kill-session> request while
      processing a confirmed commit (Section 8.4), it MUST restore the
      configuration to its state before the confirmed commit was issued.

      Otherwise, the <kill-session> operation does not roll back
      configuration or other device state modifications made by the
      entity holding the lock.

   Parameters:

      session-id:

         Session identifier of the NETCONF session to be terminated. If
         this value is equal to the current session ID, an
         "invalid-value" error is returned.

Positive Response:

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

Negative Response:

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

Example:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>4</session-id>
  </kill-session>
</rpc>

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

8. Capabilities

   This section defines a set of capabilities that a client or a server
   MAY implement. Each peer advertises its capabilities by sending them
   during an initial capabilities exchange. Each peer needs to
   understand only those capabilities that it might use and MUST ignore
   any capability received from the other peer that it does not require
   or does not understand.

   Additional capabilities can be defined using the template in
   Appendix D. Future capability definitions can be published as
   standards by standards bodies or published as proprietary extensions.

   A NETCONF capability is identified with a URI. The base capabilities
   are defined using URNs following the method described in RFC 3553
   [RFC3553]. Capabilities defined in this document have the following
   format:

      urn:ietf:params:netconf:capability:{name}:1.x

   where {name} is the name of the capability. Capabilities are often
   referenced in discussions and email using the shorthand :{name}, or
   :{name}:{version} if the capability exists in multiple versions. For
   example, the foo capability would have the formal name
   "urn:ietf:params:netconf:capability:foo:1.0" and be called ":foo".
   The shorthand form MUST NOT be used inside the protocol.

8.1. Capabilities Exchange

   Capabilities are advertised in messages sent by each peer during
   session establishment. When the NETCONF session is opened, each peer
   (both client and server) MUST send a <hello> element containing a
   list of that peer's capabilities. Each peer MUST send at least the
   base NETCONF capability, "urn:ietf:params:netconf:base:1.1". A peer
   MAY include capabilities for previous NETCONF versions, to indicate
   that it supports multiple protocol versions.

   Both NETCONF peers MUST verify that the other peer has advertised a
   common protocol version. When comparing protocol version capability
   URIs, only the base part is used, in the event any parameters are
   encoded at the end of the URI string. If no protocol version
   capability in common is found, the NETCONF peer MUST NOT continue the
   session. If more than one protocol version URI in common is present,
   then the highest numbered (most recent) protocol version MUST be used
   by both peers.

   A server sending the <hello> element MUST include a <session-id>
   element containing the session ID for this NETCONF session. A client

   sending the <hello> element MUST NOT include a <session-id> element.

   A server receiving a <hello> message with a <session-id> element MUST
   terminate the NETCONF session. Similarly, a client that does not
   receive a <session-id> element in the server's <hello> message MUST
   terminate the NETCONF session (without first sending a
   <close-session>).

   In the following example, a server advertises the base NETCONF
   capability, one NETCONF capability defined in the base NETCONF
   document, and one implementation-specific capability.

   <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <capabilities>
       <capability>
         urn:ietf:params:netconf:base:1.1
       </capability>
       <capability>
         urn:ietf:params:netconf:capability:startup:1.0
       </capability>
       <capability>
         http://example.net/router/2.3/myfeature
       </capability>
     </capabilities>
     <session-id>4</session-id>
   </hello>

   Each peer sends its <hello> element simultaneously as soon as the
   connection is open. A peer MUST NOT wait to receive the capability
   set from the other side before sending its own set.

8.2. Writable-Running Capability

8.2.1. Description

   The :writable-running capability indicates that the device supports
   direct writes to the <running> configuration datastore. In other
   words, the device supports <edit-config> and <copy-config> operations
   where the <running> configuration is the target.

8.2.2. Dependencies

   None.

8.2.3. Capability Identifier

   The :writable-running capability is identified by the following
   capability string:

    urn:ietf:params:netconf:capability:writable-running:1.0

8.2.4. New Operations

   None.

8.2.5. Modifications to Existing Operations

8.2.5.1. <edit-config>

   The :writable-running capability modifies the <edit-config> operation
   to accept the <running> element as a <target>.

8.2.5.2. <copy-config>

   The :writable-running capability modifies the <copy-config> operation
   to accept the <running> element as a <target>.

8.3. Candidate Configuration Capability

8.3.1. Description

   The candidate configuration capability, :candidate, indicates that
   the device supports a candidate configuration datastore, which is
   used to hold configuration data that can be manipulated without
   impacting the device's current configuration. The candidate
   configuration is a full configuration data set that serves as a work
   place for creating and manipulating configuration data. Additions,
   deletions, and changes can be made to this data to construct the
   desired configuration data. A <commit> operation MAY be performed at
   any time that causes the device's running configuration to be set to
   the value of the candidate configuration.

   The <commit> operation effectively sets the running configuration to
   the current contents of the candidate configuration. While it could
   be modeled as a simple copy, it is done as a distinct operation for a
   number of reasons. In keeping high-level concepts as first class
   operations, we allow developers to see more clearly both what the
   client is requesting and what the server must perform. This keeps
   the intentions more obvious, the special cases less complex, and the
   interactions between operations more straightforward. For example,
   the :confirmed-commit:1.1 capability (Section 8.4) would make no
   sense as a "copy confirmed" operation.

   The candidate configuration can be shared among multiple sessions.
   Unless a client has specific information that the candidate
   configuration is not shared, it MUST assume that other sessions are
   able to modify the candidate configuration at the same time. It is

therefore prudent for a client to lock the candidate configuration
before modifying it.

The client can discard any uncommitted changes to the candidate
configuration by executing the <discard-changes> operation. This
operation reverts the contents of the candidate configuration to the
contents of the running configuration.

8.3.2. Dependencies

None.

8.3.3. Capability Identifier

The :candidate capability is identified by the following capability
string:

      urn:ietf:params:netconf:capability:candidate:1.0

8.3.4. New Operations

8.3.4.1. <commit>

Description:

        When a candidate configuration's content is complete, the
        configuration data can be committed, publishing the data set to
        the rest of the device and requesting the device to conform to
        the behavior described in the new configuration.

        To commit the candidate configuration as the device's new
        current configuration, use the <commit> operation.

        The <commit> operation instructs the device to implement the
        configuration data contained in the candidate configuration.
        If the device is unable to commit all of the changes in the
        candidate configuration datastore, then the running
        configuration MUST remain unchanged. If the device does
        succeed in committing, the running configuration MUST be
        updated with the contents of the candidate configuration.

        If the running or candidate configuration is currently locked
        by a different session, the <commit> operation MUST fail with
        an <error-tag> value of "in-use".

        If the system does not have the :candidate capability, the
        <commit> operation is not available.

Positive Response:

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

Negative Response:

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

Example:

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

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

8.3.4.2. <discard-changes>

If the client decides that the candidate configuration is not to be
committed, the <discard-changes> operation can be used to revert the
candidate configuration to the current running configuration.

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

This operation discards any uncommitted changes by resetting the
candidate configuration with the content of the running
configuration.

8.3.5. Modifications to Existing Operations

8.3.5.1. <get-config>, <edit-config>, <copy-config>, and <validate>

The candidate configuration can be used as a source or target of any
<get-config>, <edit-config>, <copy-config>, or <validate> operation
as a <source> or <target> parameter. The <candidate> element is used
to indicate the candidate configuration:

```
   <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get-config>
       <source>
         <candidate/>
       </source>
     </get-config>
   </rpc>
```

8.3.5.2. <lock> and <unlock>

   The candidate configuration can be locked using the <lock> operation
   with the <candidate> element as the <target> parameter:

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

   Similarly, the candidate configuration is unlocked using the
   <candidate> element as the <target> parameter:

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

   When a client fails with outstanding changes to the candidate
   configuration, recovery can be difficult. To facilitate easy
   recovery, any outstanding changes are discarded when the lock is
   released, whether explicitly with the <unlock> operation or
   implicitly from session failure.

8.4. Confirmed Commit Capability

8.4.1. Description

   The :confirmed-commit:1.1 capability indicates that the server will
   support the <cancel-commit> operation and the <confirmed>,
   <confirm-timeout>, <persist>, and <persist-id> parameters for the

<commit> operation. See Section 8.3 for further details on the
<commit> operation.

A confirmed commit operation MUST be reverted if a confirming commit
is not issued within the timeout period (by default 600 seconds = 10
minutes). The confirming commit is a <commit> operation without the
<confirmed> parameter. The timeout period can be adjusted with the
<confirm-timeout> parameter. If a follow-up confirmed commit
operation is issued before the timer expires, the timer is reset to
the new value (600 seconds by default). Both the confirming commit
and a follow-up confirmed commit operation MAY introduce additional
changes to the configuration.

If the <persist> element is not given in the confirmed commit
operation, any follow-up commit and the confirming commit MUST be
issued on the same session that issued the confirmed commit. If the
<persist> element is given in the confirmed commit operation, a
follow-up commit and the confirming commit can be given on any
session, and they MUST include a <persist-id> element with a value
equal to the given value of the <persist> element.

If the server also advertises the :startup capability, a
<copy-config> from running to startup is also necessary to save the
changes to startup.

If the session issuing the confirmed commit is terminated for any
reason before the confirm timeout expires, the server MUST restore
the configuration to its state before the confirmed commit was
issued, unless the confirmed commit also included a <persist>
element.

If the device reboots for any reason before the confirm timeout
expires, the server MUST restore the configuration to its state
before the confirmed commit was issued.

If a confirming commit is not issued, the device will revert its
configuration to the state prior to the issuance of the confirmed
commit. To cancel a confirmed commit and revert changes without
waiting for the confirm timeout to expire, the client can explicitly
restore the configuration to its state before the confirmed commit
was issued, by using the <cancel-commit> operation.

For shared configurations, this feature can cause other configuration
changes (for example, via other NETCONF sessions) to be inadvertently
altered or removed, unless the configuration locking feature is used
(in other words, the lock is obtained before the <edit-config>
operation is started). Therefore, it is strongly suggested that in
order to use this feature with shared configuration datastores,

configuration locking SHOULD also be used.

Version 1.0 of this capability was defined in [RFC4741]. Version 1.1
is defined in this document, and extends version 1.0 by adding a new
operation, <cancel-commit>, and two new optional parameters,
<persist> and <persist-id>. For backwards compatibility with old
clients, servers confirming to this specification MAY advertise
version 1.0 in addition to version 1.1.

8.4.2. Dependencies

The :confirmed-commit:1.1 capability is only relevant if the
:candidate capability is also supported.

8.4.3. Capability Identifier

The :confirmed-commit:1.1 capability is identified by the following
capability string:

    urn:ietf:params:netconf:capability:confirmed-commit:1.1

8.4.4. New Operations

8.4.4.1. <cancel-commit>

Description:

    Cancels an ongoing confirmed commit. If the <persist-id>
    parameter is not given, the <cancel-commit> operation MUST be
    issued on the same session that issued the confirmed commit.

Parameters:

  persist-id:

    Cancels a persistent confirmed commit. The value MUST be
    equal to the value given in the <persist> parameter to the
    <commit> operation. If the value does not match, the
    operation fails with an "invalid-value" error.

Positive Response:

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

   Negative Response:

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

   Example:

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

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

     <rpc message-id="102"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <cancel-commit/>
     </rpc>

     <rpc-reply message-id="102"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <ok/>
     </rpc-reply>

8.4.5. Modifications to Existing Operations

8.4.5.1. <commit>

   The :confirmed-commit:1.1 capability allows 4 additional parameters
   to the <commit> operation.

   Parameters:

     confirmed:

           Perform a confirmed commit operation.

     confirm-timeout:

           Timeout period for confirmed commit, in seconds. If
           unspecified, the confirm timeout defaults to 600 seconds.

persist:

Make the confirmed commit survive a session termination, and
set a token on the ongoing confirmed commit.

persist-id:

Used to issue a follow-up confirmed commit or a confirming
commit from any session, with the token from the previous
commit operation.

Example:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <confirmed/>
    <confirm-timeout>120</confirm-timeout>
  </commit>
</rpc>

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

Example:

```
   <!-- start a persistent confirmed-commit -->
   <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <commit>
       <confirmed/>
       <persist>IQ,d4668</persist>
     </commit>
   </rpc>

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

   <!-- confirm the persistent confirmed-commit,
       possibly from another session -->
   <rpc message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <commit>
       <persist-id>IQ,d4668</persist-id>
     </commit>
   </rpc>

   <rpc-reply message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <ok/>
   </rpc-reply>
```

8.5. Rollback on Error Capability

8.5.1. Description

   This capability indicates that the server will support the
   "rollback-on-error" value in the <error-option> parameter to the
   <edit-config> operation.

   For shared configurations, this feature can cause other configuration
   changes (for example, via other NETCONF sessions) to be inadvertently
   altered or removed, unless the configuration locking feature is used
   (in other words, the lock is obtained before the <edit-config>
   operation is started). Therefore, it is strongly suggested that in
   order to use this feature with shared configuration datastores,
   configuration locking also be used.

8.5.2. Dependencies

   None

8.5.3. Capability Identifier

   The :rollback-on-error capability is identified by the following
   capability string:

      urn:ietf:params:netconf:capability:rollback-on-error:1.0

8.5.4. New Operations

   None.

8.5.5. Modifications to Existing Operations

8.5.5.1. <edit-config>

   The :rollback-on-error capability allows the "rollback-on-error"
   value to the <error-option> parameter on the <edit-config> operation.

```
   <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
       <target>
         <running/>
       </target>
       <error-option>rollback-on-error</error-option>
       <config>
         <top xmlns="http://example.com/schema/1.2/config">
           <interface>
             <name>Ethernet0/0</name>
             <mtu>100000</mtu>
           </interface>
         </top>
       </config>
     </edit-config>
   </rpc>

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

8.6. Validate Capability

8.6.1. Description

   Validation consists of checking a complete configuration for
   syntactical and semantic errors before applying the configuration to
   the device.

   If this capability is advertised, the device supports the <validate>
   protocol operation and checks at least for syntax errors. In
   addition, this capability supports the <test-option> parameter to the
   <edit-config> operation and, when it is provided, checks at least for
   syntax errors.

   Version 1.0 of this capability was defined in [RFC4741]. Version 1.1
   is defined in this document, and extends version 1.0 by adding a new
   value, "test-only", to the <test-option> parameter of the
   <edit-config> operation. For backwards compatibility with old
   clients, servers confirming to this specification MAY advertise
   version 1.0 in addition to version 1.1.

8.6.2. Dependencies

   None.

8.6.3. Capability Identifier

   The :validate:1.1 capability is identified by the following
   capability string:

      urn:ietf:params:netconf:capability:validate:1.1

8.6.4. New Operations

8.6.4.1. <validate>

   Description:

        This protocol operation validates the contents of the specified
        configuration.

   Parameters:

      source:

           Name of the configuration datastore to validate, such as
           <candidate>, or the <config> element containing the complete
           configuration to validate.

Positive Response:

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

Negative Response:

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

    A <validate> operation can fail for a number of reasons, such
    as syntax errors, missing parameters, references to undefined
    configuration data or any other violations of rules established
    by the underlying data model.

Example:

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

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

8.6.5. Modifications to Existing Operations

8.6.5.1. <edit-config>

The :validate:1.1 capability modifies the <edit-config> operation to
accept the <test-option> parameter.

8.7. Distinct Startup Capability

8.7.1. Description

The device supports separate running and startup configuration
datastores. The startup configuration is loaded by the device when
it boots. Operations that affect the running configuration will not
be automatically copied to the startup configuration. An explicit
<copy-config> operation from the <running> to the <startup> is used
to update the startup configuration to the current contents of the

running configuration. NETCONF protocol operations refer to the
startup datastore using the <startup> element.

8.7.2. Dependencies

None.

8.7.3. Capability Identifier

The :startup capability is identified by the following capability
string:

    urn:ietf:params:netconf:capability:startup:1.0

8.7.4. New Operations

None.

8.7.5. Modifications to Existing Operations

8.7.5.1. General

The :startup capability adds the <startup/> configuration datastore
to arguments of several NETCONF operations. The server MUST support
the following additional values:

Operation	Parameters	Notes
<get-config>	<source>	
<copy-config>	<source> <target>	
<lock>	<target>	
<unlock>	<target>	
<validate>	<source>	If :validate:1.1 is advertised
<delete-config>	<target>	Resets the device to its factory defaults

To save the startup configuration, use the <copy-config> operation to
copy the <running> configuration datastore to the <startup>
configuration datastore.

```
    <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <copy-config>
        <target>
          <startup/>
        </target>
        <source>
          <running/>
        </source>
      </copy-config>
    </rpc>
```

8.8. URL Capability

8.8.1. Description

   The NETCONF peer has the ability to accept the <url> element in
   <source> and <target> parameters. The capability is further
   identified by URL arguments indicating the URL schemes supported.

8.8.2. Dependencies

   None.

8.8.3. Capability Identifier

   The :url capability is identified by the following capability string:

   urn:ietf:params:netconf:capability:url:1.0?scheme={name,...}

   The :url capability URI MUST contain a "scheme" argument assigned a
   comma-separated list of scheme names indicating which schemes the
   NETCONF peer supports. For example:

      urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file

8.8.4. New Operations

   None.

8.8.5. Modifications to Existing Operations

8.8.5.1. <edit-config>

   The :url capability modifies the <edit-config> operation to accept
   the <url> element as an alternative to the <config> parameter.

   The file that the url refers to contains the configuration data

   hierarchy to be modified, encoded in XML under the element <config>
   in the "urn:ietf:params:xml:ns:netconf:base:1.0" namespace.

8.8.5.2. <copy-config>

   The :url capability modifies the <copy-config> operation to accept
   the <url> element as the value of the <source> and the <target>
   parameters.

   The file that the url refers to contains the complete datastore,
   encoded in XML under the element <config> in the
   "urn:ietf:params:xml:ns:netconf:base:1.0" namespace.

8.8.5.3. <delete-config>

   The :url capability modifies the <delete-config> operation to accept
   the <url> element as the value of the <target> parameters.

8.8.5.4. <validate>

   The :url capability modifies the <validate> operation to accept the
   <url> element as the value of the <source> parameter.

8.9. XPath Capability

8.9.1. Description

   The XPath capability indicates that the NETCONF peer supports the use
   of XPath expressions in the <filter> element. XPath is described in
   [W3C.REC-xpath-19991116].

   The data model used in the XPath expression is the same as that used
   in XPath 1.0 [W3C.REC-xpath-19991116], with the same extension for
   root node children as used by XSLT 1.0 [W3C.REC-xslt-19991116]
   (section 3.1). Specifically, it means that the root node MAY have
   any number of element nodes as its children.

   The XPath expression is evaluated in the following context:

   o The set of namespace declarations are those in scope on the
      <filter> element.

   o The set of variable bindings is defined by the data model. If no
      such variable bindings are defined, the set is empty.

   o The function library is the core function library, plus any
      functions defined by the data model.

   o The context node is the root node.

   The XPath expression MUST return a node set. If it does not return a
   node set, the operation fails with an "invalid-value" error.

   The response message contains the subtrees selected by the filter
   expression. For each such subtree, the path from the data model root
   node down to the subtree, including any elements or attributes
   necessary to uniquely identify the subtree, are included in the
   response message. Specific data instances are not duplicated in the
   respone.

8.9.2. Dependencies

   None.

8.9.3. Capability Identifier

   The :xpath capability is identified by the following capability
   string:

      urn:ietf:params:netconf:capability:xpath:1.0

8.9.4. New Operations

   None.

8.9.5. Modifications to Existing Operations

8.9.5.1. <get-config> and <get>

   The :xpath capability modifies the <get> and <get-config> operations
   to accept the value "xpath" in the "type" attribute of the <filter>
   element. When the "type" attribute is set to "xpath", a "select"
   attribute MUST be present on the <filter> element. The "select"
   attribute will be treated as an XPath expression and used to filter
   the returned data. The <filter> element itself MUST be empty in this
   case.

   The XPath result for the select expression MUST be a node-set. Each
   node in the node-set MUST correspond to a node in underlying data
   model. In order to properly identify each node, the following
   encoding rules are defined:

   o All ancestor nodes of the result node MUST be encoded first, so
      the <data> element returned in the reply contains only fully-
      specified sub-trees, according to the underlying data model.

   o If any sibling or ancestor nodes of the result node are needed to
      identify a particular instance within a conceptual data structure,
      then these nodes MUST also be encoded in the response.

   For example:

```
  <rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
      <source>
        <running/>
      </source>
      <!-- get the user named fred -->
      <filter xmlns:t="http://example.com/schema/1.2/config"
              type="xpath"
              select="/t:top/t:users/t:user[t:name='fred']"/>
    </get-config>
  </rpc>

  <rpc-reply message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user>
            <name>fred</name>
            <company-info>
              <id>2</id>
            </company-info>
          </user>
        </users>
      </top>
    </data>
  </rpc-reply>
```

9. Security Considerations

   This section provides security considerations for the base NETCONF
   message layer and the base operations of the NETCONF protocol.
   Security considerations for the NETCONF transports are provided in
   the transport documents and security considerations for the content
   manipulated by NETCONF can be found in the documents defining data
   models.

   This document does not specify an authorization scheme, as such a
   scheme will likely be tied to a meta-data model or a data model.
   Implementors SHOULD provide a comprehensive authorization scheme with
   NETCONF.

   Authorization of individual users via the NETCONF server may or may
   not map 1:1 to other interfaces. First, the data models might be
   incompatible. Second, it could be desirable to authorize based on
   mechanisms available in the secure transport layer (SSH, BEEP, etc).

   In addition, operations on configurations could have unintended
   consequences if those operations are also not guarded by the global
   lock on the files or objects being operated upon. For instance, a
   partially complete access list could be committed from a candidate
   configuration unbeknownst to the owner of the lock of the candidate
   configuration, leading to either an insecure or inaccessible device
   if the lock on the candidate configuration does not also apply to the
   <copy-config> operation when applied to it.

   Configuration information is by its very nature sensitive. Its
   transmission in the clear and without integrity checking leaves
   devices open to classic eavesdropping and false data injection
   attacks. Configuration information often contains passwords, user
   names, service descriptions, and topological information, all of
   which are sensitive. Because of this, this protocol SHOULD be
   implemented carefully with adequate attention to all manner of attack
   one might expect to experience with other management interfaces.

   The protocol, therefore, MUST minimally support options for both
   confidentiality and authentication. It is anticipated that the
   underlying protocol (SSH, BEEP, etc) will provide for both
   confidentiality and authentication, as is required. It is further
   expected that the identity of each end of a NETCONF session will be
   available to the other in order to determine authorization for any
   given request. One could also easily envision additional
   information, such as transport and encryption methods, being made
   available for purposes of authorization. NETCONF itself provide no
   means to re-authenticate, much less authenticate. All such actions
   occur at lower layers.

Different environments may well allow different rights prior to and
then after authentication. Thus, an authorization model is not
specified in this document. When an operation is not properly
authorized, a simple "access denied" is sufficient. Note that
authorization information can be exchanged in the form of
configuration information, which is all the more reason to ensure the
security of the connection.

That having been said, it is important to recognize that some
operations are clearly more sensitive by nature than others. For
instance, <copy-config> to the startup or running configurations is
clearly not a normal provisioning operation, whereas <edit-config>
is. Such global operations MUST disallow the changing of information
that an individual does not have authorization to perform. For
example, if a user A is not allowed to configure an IP address on an
interface but user B has configured an IP address on an interface in
the <candidate> configuration, user A MUST NOT be allowed to commit
the <candidate> configuration.

Similarly, just because someone says "go write a configuration
through the URL capability at a particular place", this does not mean
that an element will do it without proper authorization.

The <lock> operation will demonstrate that NETCONF is intended for
use by systems that have at least some trust of the administrator.
As specified in this document, it is possible to lock portions of a
configuration that a principal might not otherwise have access to.
After all, the entire configuration is locked. To mitigate this
problem, there are two approaches. It is possible to kill another
NETCONF session programmatically from within NETCONF if one knows the
session identifier of the offending session. The other possible way
to break a lock is to provide an function within the device's native
user interface. These two mechanisms suffer from a race condition
that could be ameliorated by removing the offending user from an AAA
server. However, such a solution is not useful in all deployment
scenarios, such as those where SSH public/private key pairs are used.

10. IANA Considerations

10.1. NETCONF XML Namespace

   This document registers a URI for the NETCONF XML namespace in the
   IETF XML registry [RFC3688].

   IANA is requested to update the allocation of the following URI to
   reference this document when it is published as an RFC.

   URI: urn:ietf:params:xml:ns:netconf:base:1.0

   Registrant Contact: The IESG.

   XML: N/A, the requested URI is an XML namespace.

10.2. NETCONF XML Schema

   This document registers a URI for the NETCONF XML schema in the IETF
   XML registry [RFC3688].

   IANA is requested to update the allocation of the following URI to
   reference this document when it is published as an RFC.

   URI: urn:ietf:params:xml:schema:netconf

   Registrant Contact: The IESG.

   XML: Appendix B of this document.

10.3. NETCONF YANG Module

   This document registers a YANG module in the YANG Module Names
   registry [RFC6020].

      name: ietf-netconf
      namespace: urn:ietf:params:xml:ns:netconf:base:1.0
      prefix: nc
      reference: RFCXXXX

10.4. NETCONF Capability URNs

   IANA has created and will maintain a registry "Network Configuration
   Protocol (NETCONF) Capability URNs" that allocates NETCONF capability
   identifiers. Additions to the registry require IETF Standards
   Action.

   IANA is requested to update the allocations of the following

capabilities to reference this document when it is published as an
RFC.

```
+--------------------+--------------------------------------------+
| Index              | Capability Identifier                      |
+--------------------+--------------------------------------------+
| :writable-running  | urn:ietf:params:netconf:capability:writable- |
|                    | running:1.0                                |
|                    |                                            |
| :candidate         | urn:ietf:params:netconf:capability:candidate |
|                    | :1.0                                       |
|                    |                                            |
| :rollback-on-error | urn:ietf:params:netconf:capability:rollback- |
|                    | on-error:1.0                               |
|                    |                                            |
| :startup           | urn:ietf:params:netconf:capability:startup:1 |
|                    | .0                                         |
|                    |                                            |
| :url               | urn:ietf:params:netconf:capability:url:1.0 |
|                    |                                            |
| :xpath             | urn:ietf:params:netconf:capability:xpath:1.0 |
+--------------------+--------------------------------------------+
```

IANA is requested to add the following capabilities to the registry:

```
+--------------------+--------------------------------------------+
| Index              | Capability Identifier                      |
+--------------------+--------------------------------------------+
| :base:1.1          | urn:ietf:params:netconf:base:1.1           |
|                    |                                            |
| :confirmed-commit:1 | urn:ietf:params:netconf:capability:confirme |
| .1                 | d-commit:1.1                               |
|                    |                                            |
| :validate:1.1      | urn:ietf:params:netconf:capability:validate |
|                    | :1.1                                       |
+--------------------+--------------------------------------------+
```

11. Authors and Acknowledgements

   This document was written by:

      Andy Bierman

      Ken Crozier, Cisco Systems

      Rob Enns, Juniper Networks

      Ted Goddard, IceSoft

      Eliot Lear, Cisco Systems

      Phil Shafer, Juniper Networks

      Steve Waldbusser

      Margaret Wasserman, ThingMagic

   The authors would like to acknowledge the members of the NETCONF
   working group. In particular, we would like to thank Wes Hardaker
   for his persistence and patience in assisting us with security
   considerations. We would also like to thank Randy Presuhn, Sharon
   Chisholm, Glenn Waters, David Perkins, Weijing Chen, Simon Leinen,
   Keith Allen, Dave Harrington, Ladislav Lhotka, Tom Petch, and Kent
   Watsen for all of their valuable advice.

12. References

12.1. Normative References

   [W3C.REC-xml-20001006]
             Bray, T., Sperberg-McQueen, C., Paoli, J., and E. Maler,
             "Extensible Markup Language (XML) 1.0 (Second Edition)",
             World Wide Web Consortium FirstEdition REC-xml-20001006,
             October 2000,
             <http://www.w3.org/TR/2000/REC-xml-20001006>.

   [W3C.REC-xpath-19991116]
             DeRose, S. and J. Clark, "XML Path Language (XPath)
             Version 1.0", World Wide Web Consortium
             Recommendation REC-xpath-19991116, November 1999,
             <http://www.w3.org/TR/1999/REC-xpath-19991116>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
             Requirement Levels", BCP 14, RFC 2119, March 1997.

   [I-D.ietf-netconf-rfc4742bis]
             Wasserman, M. and T. Goddard, "Using the NETCONF
             Configuration Protocol over Secure Shell (SSH)",
             draft-ietf-netconf-rfc4742bis-07 (work in progress),
             February 2011.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
             Resource Identifier (URI): Generic Syntax", STD 66,
             RFC 3986, January 2005.

   [RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An
             IETF URN Sub-namespace for Registered Protocol
             Parameters", BCP 73, RFC 3553, June 2003.

   [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
             10646", STD 63, RFC 3629, November 2003.

   [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
             January 2004.

   [RFC5717] Lengyel, B. and M. Bjorklund, "Partial Lock Remote
             Procedure Call (RPC) for NETCONF", RFC 5717,
             December 2009.

   [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the
             Network Configuration Protocol (NETCONF)", RFC 6020,
             October 2010.

   [RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021,
              October 2010.

12.2. Informative References

   [W3C.REC-xslt-19991116]
              Clark, J., "XSL Transformations (XSLT) Version 1.0", World
              Wide Web Consortium Recommendation REC-xslt-19991116,
              November 1999,
              <http://www.w3.org/TR/1999/REC-xslt-19991116>.

   [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson,
              "Remote Authentication Dial In User Service (RADIUS)",
              RFC 2865, June 2000.

   [RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for
              the Use of Extensible Markup Language (XML)
              within IETF Protocols", BCP 70, RFC 3470, January 2003.

   [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH)
              Protocol Architecture", RFC 4251, January 2006.

   [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741,
              December 2006.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

Appendix A. NETCONF Error List

   This section is normative.

   For each error-tag, the valid error-type and error-severity values
   are listed, together with any mandatory error-info, if any.

```
error-tag:      in-use
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    The request requires a resource that already is in
                use.

error-tag:      invalid-value
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    The request specifies an unacceptable value for one
                or more parameters.

error-tag:      too-big
error-type:     transport, rpc, protocol, application
error-severity: error
error-info:     none
Description:    The request or response (that would be generated) is
                too large for the implementation to handle.

error-tag:      missing-attribute
error-type:     rpc, protocol, application
error-severity: error
error-info:     <bad-attribute> : name of the missing attribute
                <bad-element> : name of the element that is supposed to
                  contain the missing attribute
Description:    An expected attribute is missing.

error-tag:      bad-attribute
error-type:     rpc, protocol, application
error-severity: error
error-info:     <bad-attribute> : name of the attribute w/ bad value
                <bad-element> : name of the element that contains
                  the attribute with the bad value
Description:    An attribute value is not correct; e.g., wrong type,
                out of range, pattern mismatch.

error-tag:      unknown-attribute
error-type:     rpc, protocol, application
error-severity: error
```

```
error-info:     <bad-attribute> : name of the unexpected attribute
                <bad-element> : name of the element that contains
                  the unexpected attribute
Description:    An unexpected attribute is present.


error-tag:      missing-element
error-type:     protocol, application
error-severity: error
error-info:     <bad-element> : name of the missing element
Description:    An expected element is missing.


error-tag:      bad-element
error-type:     protocol, application
error-severity: error
error-info:     <bad-element> : name of the element w/ bad value
Description:    An element value is not correct; e.g., wrong type,
                out of range, pattern mismatch.


error-tag:      unknown-element
error-type:     protocol, application
error-severity: error
error-info:     <bad-element> : name of the unexpected element
Description:    An unexpected element is present.


error-tag:      unknown-namespace
error-type:     protocol, application
error-severity: error
error-info:     <bad-element> : name of the element that contains
                  the unexpected namespace
                <bad-namespace> : name of the unexpected namespace
Description:    An unexpected namespace is present.


error-tag:      access-denied
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    Access to the requested protocol operation, or
                data model is denied because authorization failed.


error-tag:      lock-denied
error-type:     protocol
error-severity: error
error-info:     <session-id> : session ID of session holding the
                  requested lock, or zero to indicate a non-NETCONF
                  entity holds the lock
Description: Access to the requested lock is denied because the
            lock is currently held by another entity.
```

```
error-tag:       resource-denied
error-type:      transport, rpc, protocol, application
error-severity:  error
error-info:      none
Description:      Request could not be completed because of
                 insufficient resources.


error-tag:       rollback-failed
error-type:      protocol, application
error-severity:  error
error-info:      none
Description:      Request to rollback some configuration change (via
                 rollback-on-error or discard-changes operations) was
                 not completed for some reason.


error-tag:       data-exists
error-type:      application
error-severity:  error
error-info:      none
Description:      Request could not be completed because the relevant
                 data model content already exists. For example,
                 a "create" operation was attempted on data that
                 already exists.


error-tag:       data-missing
error-type:      application
error-severity:  error
error-info:      none
Description:      Request could not be completed because the relevant
                 data model content does not exist.  For example,
                 a "delete" operation was attempted on
                 data that does not exist.


error-tag:       operation-not-supported
error-type:      protocol, application
error-severity:  error
error-info:      none
Description:      Request could not be completed because the requested
                 operation is not supported by this implementation.


error-tag:       operation-failed
error-type:      rpc, protocol, application
error-severity:  error
error-info:      none
Description:      Request could not be completed because the requested
                 operation failed for some reason not covered by
                 any other error condition.
```

```
error-tag:      partial-operation
error-type:     application
error-severity: error
error-info:     <ok-element> : identifies an element in the data
                  model for which the requested operation has been
                  completed for that node and all its child nodes.
                  This element can appear zero or more times in the
                  <error-info> container.

                <err-element> : identifies an element in the data
                  model for which the requested operation has failed
                  for that node and all its child nodes.
                  This element can appear zero or more times in the
                  <error-info> container.

                <noop-element> : identifies an element in the data
                  model for which the requested operation was not
                  attempted for that node and all its child nodes.
                  This element can appear zero or more times in the
                  <error-info> container.

 Description:    This error-tag is obsolete, and SHOULD NOT be sent
                 by servers conforming to this document.

                 Some part of the requested operation failed or was
                 not attempted for some reason.  Full cleanup has
                 not been performed (e.g., rollback not supported)
                 by the server.  The error-info container is used
                 to identify which portions of the application
                 data model content for which the requested operation
                 has succeeded (<ok-element>), failed (<bad-element>),
                 or not been attempted (<noop-element>).

 error-tag:      malformed-message
 error-type:     rpc
 error-severity: error
 error-info:     none
 Description:    A message could not be handled because it failed to
                 be parsed correctly. For example, the message is not
                 well-formed XML or it uses an invalid character set.

                 This error-tag is new in :base:1.1 and MUST NOT be
                 sent to old clients.
```

Appendix B. XML Schema for NETCONF Messages Layer

   This section is normative.

   <CODE BEGINS> file "netconf.xsd"

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           targetNamespace="urn:ietf:params:xml:ns:netconf:base:1.0"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xml:lang="en"
           version="1.1">

  <xs:annotation>
    <xs:documentation>
      This schema defines the syntax for the NETCONF Messages layer
      messages 'hello', 'rpc', and 'rpc-reply'.
    </xs:documentation>
  </xs:annotation>

  <!--
      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>
  <!--
      message-id attribute
    -->
  <xs:simpleType name="messageIdType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="4095"/>
    </xs:restriction>
  </xs:simpleType>
  <!--
      Types used for session-id
    -->
  <xs:simpleType name="SessionId">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1"/>
```

```
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="SessionIdOrZero">
      <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <!--
       <rpc> element
      -->
    <xs:complexType name="rpcType">
      <xs:sequence>
        <xs:element ref="rpcOperation"/>
      </xs:sequence>
      <xs:attribute name="message-id" type="messageIdType"
                    use="required"/>
      <!--
         Arbitrary attributes can be supplied with <rpc> element.
        -->
      <xs:anyAttribute processContents="lax"/>
    </xs:complexType>
    <xs:element name="rpc" type="rpcType"/>
    <!--
       data types and elements used to construct rpc-errors
      -->
    <xs:simpleType name="ErrorType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="transport"/>
        <xs:enumeration value="rpc"/>
        <xs:enumeration value="protocol"/>
        <xs:enumeration value="application"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ErrorTag">
      <xs:restriction base="xs:string">
        <xs:enumeration value="in-use"/>
        <xs:enumeration value="invalid-value"/>
        <xs:enumeration value="too-big"/>
        <xs:enumeration value="missing-attribute"/>
        <xs:enumeration value="bad-attribute"/>
        <xs:enumeration value="unknown-attribute"/>
        <xs:enumeration value="missing-element"/>
        <xs:enumeration value="bad-element"/>
        <xs:enumeration value="unknown-element"/>
        <xs:enumeration value="unknown-namespace"/>
        <xs:enumeration value="access-denied"/>
        <xs:enumeration value="lock-denied"/>
        <xs:enumeration value="resource-denied"/>
        <xs:enumeration value="rollback-failed"/>
        <xs:enumeration value="data-exists"/>
```

```
        <xs:enumeration value="data-missing"/>
        <xs:enumeration value="operation-not-supported"/>
        <xs:enumeration value="operation-failed"/>
        <xs:enumeration value="partial-operation"/>
        <xs:enumeration value="malformed-message"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ErrorSeverity">
      <xs:restriction base="xs:string">
        <xs:enumeration value="error"/>
        <xs:enumeration value="warning"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="errorInfoType">
      <xs:sequence>
        <xs:choice>
          <xs:element name="session-id" type="SessionIdOrZero"/>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:sequence>
              <xs:element name="bad-attribute" type="xs:QName"
                          minOccurs="0" maxOccurs="1"/>
              <xs:element name="bad-element" type="xs:QName"
                          minOccurs="0" maxOccurs="1"/>
              <xs:element name="ok-element" type="xs:QName"
                          minOccurs="0" maxOccurs="1"/>
              <xs:element name="err-element" type="xs:QName"
                          minOccurs="0" maxOccurs="1"/>
              <xs:element name="noop-element" type="xs:QName"
                          minOccurs="0" maxOccurs="1"/>
              <xs:element name="bad-namespace" type="xs:string"
                          minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
          </xs:sequence>
        </xs:choice>
        <!-- elements from any other namespace are also allowed
             to follow the NETCONF elements -->
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rpcErrorType">
      <xs:sequence>
        <xs:element name="error-type" type="ErrorType"/>
        <xs:element name="error-tag" type="ErrorTag"/>
        <xs:element name="error-severity" type="ErrorSeverity"/>
        <xs:element name="error-app-tag" type="xs:string"
                    minOccurs="0"/>
        <xs:element name="error-path" type="xs:string" minOccurs="0"/>
```

```
      <xs:element name="error-message" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute ref="xml:lang" use="optional"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="error-info" type="errorInfoType"
                  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!--
     operation attribute used in <edit-config>
     -->
  <xs:simpleType name="editOperationType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="merge"/>
      <xs:enumeration value="replace"/>
      <xs:enumeration value="create"/>
      <xs:enumeration value="delete"/>
      <xs:enumeration value="remove"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:attribute name="operation" type="editOperationType"/>
  <!--
     <rpc-reply> element
     -->
  <xs:complexType name="rpcReplyType">
    <xs:choice>
      <xs:element name="ok"/>
      <xs:sequence>
        <xs:element ref="rpc-error"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="rpcResponse"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:choice>
    <xs:attribute name="message-id" type="messageIdType"
                  use="optional"/>
    <!--
       Any attributes supplied with <rpc> element must be returned
       on <rpc-reply>.
       -->
    <xs:anyAttribute processContents="lax"/>
  </xs:complexType>
  <xs:element name="rpc-reply" type="rpcReplyType"/>
```

```
   <!--
      <rpc-error> element
        -->
   <xs:element name="rpc-error" type="rpcErrorType"/>
   <!--
      rpcOperationType: used as a base type for all
      NETCONF operations
     -->
   <xs:complexType name="rpcOperationType"/>
   <xs:element name="rpcOperation" type="rpcOperationType"
             abstract="true"/>
   <!--
      rpcResponseType: used as a base type for all
      NETCONF responses
     -->
   <xs:complexType name="rpcResponseType"/>
   <xs:element name="rpcResponse" type="rpcResponseType"
             abstract="true"/>
   <!--
      <hello> element
     -->
   <xs:element name="hello">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="capabilities">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="capability" type="xs:anyURI"
                         maxOccurs="unbounded"/>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="session-id" type="SessionId"
                   minOccurs="0"/>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
  </xs:schema>


  <CODE ENDS>
```

Appendix C.   YANG Module for NETCONF Protocol Operations

   This section is normative.

   The ietf-netconf YANG module imports typedefs from [RFC6021].

 // RFC Ed.: please update the date to the date of publication
 <CODE BEGINS> file "ietf-netconf@2011-03-08.yang"

 module ietf-netconf {

   // the namespace for NETCONF XML definitions is unchanged
   // from RFC 4741 which this document replaces
   namespace "urn:ietf:params:xml:ns:netconf:base:1.0";

   prefix nc;

   import ietf-inet-types {
     prefix inet;
   }

   organization
     "IETF NETCONF (Network Configuration) Working Group";

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

      WG Chair: Bert Wijnen
                <mailto:bertietf@bwijnen.net>

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

      Editor: Martin Bjorklund
                <mailto:mbj@tail-f.com>

      Editor: Juergen Schoenwaelder
                <mailto:j.schoenwaelder@jacobs-university.de>

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

   description
     "NETCONF Protocol Data Types and Protocol Operations.

      Copyright (c) 2010 IETF Trust and the persons identified as
      the document authors. All rights reserved.

          Redistribution and use in source and binary forms, with or
          without modification, is permitted pursuant to, and subject
          to the license terms contained in, the Simplified BSD License
          set forth in Section 4.c of the IETF Trust's Legal Provisions
          Relating to IETF Documents
          (http://trustee.ietf.org/license-info).

          This version of this YANG module is part of RFC XXXX; see
          the RFC itself for full legal notices.";
       // RFC Ed.: replace XXXX with actual RFC number and remove this note

      // RFC Ed.: please update the date to the date of publication
      revision 2011-03-08 {
        description
          "Initial revision";
        reference
          "RFC XXXX: Network Configuration Protocol";
      }

      extension get-filter-element-attributes {
        description
          "If this extension is present within the
           an 'anyxml' statement named 'filter', which must be
           conceptually defined within the RPC input section
           for the 'get' and 'get-config' protocol operations,
           then the following unqualified XML attribute is
           supported within the 'filter' element, within
           a 'get' or 'get-config' protocol operation:

             type : optional attribute with allowed
                     value strings 'subtree' and 'xpath'.
                     If missing, the default value is 'subtree'.

           If the 'xpath' feature is supported, then the
           following unqualified XML attribute is
           also supported:

             select: optional attribute containing a
                     string representing an XPath expression.
                     The 'type' attribute must be equal to 'xpath'
                     if this attribute is present.";
      }

      // NETCONF capabilities defined as features
      feature writable-running {
        description
          "NETCONF :writable-running capability;
           If the server advertises the :writable-running

```
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.2.";
   }

   feature candidate {
     description
       "NETCONF :candidate capability;
        If the server advertises the :candidate
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.3.";
   }

   feature confirmed-commit {
     if-feature candidate;
     description
       "NETCONF :confirmed-commit:1.1 capability;
        If the server advertises the :confirmed-commit:1.1
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";

     reference "RFC XXXX, section 8.4.";
   }

   feature rollback-on-error {
     description
       "NETCONF :rollback-on-error capability;
        If the server advertises the :rollback-on-error
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.5.";
   }

   feature validate {
     description
       "NETCONF :validate:1.1 capability;
        If the server advertises the :validate:1.1
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.6.";
   }
```

```
   feature startup {
     description
       "NETCONF :startup capability;
        If the server advertises the :startup
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.7.";
   }

   feature url {
     description
       "NETCONF :url capability;
        If the server advertises the :url
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.8.";
   }

   feature xpath {
     description
       "NETCONF :xpath capability;
        If the server advertises the :xpath
        capability for a session, then this feature must
        also be enabled for that session.  Otherwise,
        this feature must not be enabled.";
     reference "RFC XXXX, section 8.9.";
   }

   // NETCONF Simple Types

   typedef session-id-type {
     type uint32 {
       range "1..max";
     }
     description
       "NETCONF Session Id";
   }

   typedef session-id-or-zero-type {
     type uint32;
     description
       "NETCONF Session Id or Zero to indicate none";
   }

   typedef error-tag-type {
     type enumeration {
```

```
        enum in-use {
          description
            "The request requires a resource that
             already is in use.";
        }
        enum invalid-value {
          description
            "The request specifies an unacceptable value for one
             or more parameters.";
        }
        enum too-big {
          description
            "The request or response (that would be generated) is
             too large for the implementation to handle.";
        }
        enum missing-attribute {
          description
            "An expected attribute is missing.";
        }
        enum bad-attribute {
          description
            "An attribute value is not correct; e.g., wrong type,
             out of range, pattern mismatch.";
        }
        enum unknown-attribute {
          description
            "An unexpected attribute is present.";
        }
        enum missing-element {
          description
            "An expected element is missing.";
        }
        enum bad-element {
          description
            "An element value is not correct; e.g., wrong type,
             out of range, pattern mismatch.";
        }
        enum unknown-element {
          description
            "An unexpected element is present.";
        }
        enum unknown-namespace {
          description
            "An unexpected namespace is present.";
        }
        enum access-denied {
          description
            "Access to the requested protocol operation, or
```

```
            data model is denied because authorization failed.";
        }
        enum lock-denied {
          description
            "Access to the requested lock is denied because the
             lock is currently held by another entity.";
        }
        enum resource-denied {
          description
            "Request could not be completed because of
             insufficient resources.";
        }
        enum rollback-failed {
          description
            "Request to rollback some configuration change (via
             rollback-on-error or discard-changes operations) was
             not completed for some reason.";

        }
        enum data-exists {
          description
            "Request could not be completed because the relevant
             data model content already exists. For example,
             a 'create' operation was attempted on data that
             already exists.";
        }
        enum data-missing {
          description
            "Request could not be completed because the relevant
             data model content does not exist.  For example,
             a 'delete' operation was attempted on
             data that does not exist.";
        }
        enum operation-not-supported {
          description
            "Request could not be completed because the requested
             operation is not supported by this implementation.";
        }
        enum operation-failed {
          description
            "Request could not be completed because the requested
             operation failed for some reason not covered by
             any other error condition.";
        }
        enum partial-operation {
          description
            "This error-tag is obsolete, and SHOULD NOT be sent
             by servers conforming to this document.";
```

```
        }
        enum malformed-message {
          description
            "A message could not be handled because it failed to
             be parsed correctly. For example, the message is not
             well-formed XML or it uses an invalid character set.";
        }
      }
      description "NETCONF Error Tag";
      reference "RFC XXXX, Appendix A.";
    }

    typedef error-severity-type {
      type enumeration {
        enum error {
          description "Error severity";
        }
        enum warning {
          description "Warning severity";
        }
      }
      description "NETCONF Error Severity";
      reference "RFC XXXX, section 4.3.";
    }

    typedef edit-operation-type {
      type enumeration {
        enum merge {
          description
            "The configuration data identified by the
             element containing this attribute is merged
             with the configuration at the corresponding
             level in the configuration datastore identified
             by the target parameter.";
        }
        enum replace {
          description
            "The configuration data identified by the element
             containing this attribute replaces any related
             configuration in the configuration datastore
             identified by the target parameter.  If no such
             configuration data exists in the configuration
             datastore, it is created. Unlike a
             <copy-config> operation, which replaces the
             entire target configuration, only the configuration
             actually present in the config parameter is affected.";
        }
        enum create {
```

```
        description
          "The configuration data identified by the element
           containing this attribute is added to the
           configuration if and only if the configuration
           data does not already exist in the configuration
           datastore.  If the configuration data exists, an
           <rpc-error> element is returned with an
           <error-tag> value of 'data-exists'.";
      }
      enum delete {
        description
          "The configuration data identified by the element
           containing this attribute is deleted from the
           configuration if and only if the configuration
           data currently exists in the configuration
           datastore.  If the configuration data does not
           exist, an <rpc-error> element is returned with
           an <error-tag> value of 'data-missing'.";
      }
      enum remove {
        description
          "The configuration data identified by the element
           containing this attribute is deleted from the
           configuration if the configuration
           data currently exists in the configuration
           datastore.  If the configuration data does not
           exist, the 'remove' operation is silently ignored
           by the server.";
      }
    }
    default "merge";
    description "NETCONF 'operation' attribute values";
    reference "RFC XXXX, section 7.2.";
  }

  // NETCONF Standard Protocol Operations

  rpc get-config {
    description
      "Retrieve all or part of a specified configuration.";

    reference "RFC XXXX, section 7.1.";

    input {
      container source {
        description
          "Particular configuration to retrieve.";
```

```
      choice config-source {
        mandatory true;
        description
          "The configuration to retrieve.";
        leaf candidate {
          if-feature candidate;
          type empty;
          description
            "The candidate configuration is the config source.";
        }
        leaf running {
          type empty;
          description
            "The running configuration is the config source.";
        }
        leaf startup {
          if-feature startup;
          type empty;
          description
            "The startup configuration is the config source.
             This is optional-to-implement on the server because
             not all servers will support filtering for this
             datastore.";
        }
      }
    }

    anyxml filter {
      description
        "Subtree or XPath filter to use.";
      nc:get-filter-element-attributes;
    }
  }

  output {
    anyxml data {
      description
        "Copy of the source datastore subset which matched
         the filter criteria (if any).  An empty data container
         indicates that the request did not produce any results.";
    }
  }
}

rpc edit-config {
  description
    "The 'edit-config' operation loads all or part of a specified
     configuration to the specified target configuration.";
```

```
     reference "RFC XXXX, section 7.2.";

     input {
       container target {
         description
           "Particular configuration to edit.";

         choice config-target {
           mandatory true;
           description
             "The configuration target.";

           leaf candidate {
             if-feature candidate;
             type empty;
             description
               "The candidate configuration is the config target.";
           }
           leaf running {
             if-feature writable-running;
             type empty;
             description
               "The running configuration is the config source.";
           }
         }
       }

       leaf default-operation {
         type enumeration {
           enum merge {
             description
               "The default operation is merge.";
           }
           enum replace {
             description
               "The default operation is replace.";
           }
           enum none {
             description
               "There is no default operation.";
           }
         }
         default "merge";
         description
           "The default operation to use.";
       }

       leaf test-option {
```

```
         if-feature validate;
         type enumeration {
           enum test-then-set {
             description
               "The server will test and then set if no errors.";
           }
           enum set {
             description
               "The server will set without a test first.";
           }

           enum test-only {
             description
               "The server will only test and not set, even
                if there are no errors.";
           }
         }
         default "test-then-set";
         description
           "The test option to use.";
       }

       leaf error-option {
         type enumeration {
           enum stop-on-error {
             description
               "The server will stop on errors.";
           }
           enum continue-on-error {
             description
               "The server may continue on errors.";
           }
           enum rollback-on-error {
             description
               "The server will rollback on errors.
                This value can only be used if the 'rollback-on-error'
                feature is supported.";
           }
         }
         default "stop-on-error";
         description
           "The error option to use.";
       }

       choice edit-content {
         mandatory true;
         description
           "The content for the edit operation";
```

```
        anyxml config {
          description
            "Inline Config content.";
        }
        leaf url {
          if-feature url;
          type inet:uri;
          description
            "URL based config content.";
        }
      }
    }
  }

rpc copy-config {
  description
    "Create or replace an entire configuration datastore with the
     contents of another complete configuration datastore.";

  reference "RFC XXXX, section 7.3.";

  input {
    container target {
      description
        "Particular configuration to copy to.";

      choice config-target {
        mandatory true;
        description
          "The configuration target of the copy operation.";

        leaf candidate {
          if-feature candidate;
          type empty;
          description
            "The candidate configuration is the config target.";
        }
        leaf running {
          if-feature writable-running;
          type empty;
          description
            "The running configuration is the config target.
             This is optional-to-implement on the server.";
        }
        leaf startup {
          if-feature startup;
          type empty;
          description
```

```
              "The startup configuration is the config target.";
          }
          leaf url {
            if-feature url;
            type inet:uri;
            description
              "The URL-based configuration is the config target.";
          }
        }
      }
    }

    container source {
      description
        "Particular configuration to copy from.";

      choice config-source {
        mandatory true;
        description
          "The configuration source for the copy operation.";

        leaf candidate {
          if-feature candidate;
          type empty;
          description
            "The candidate configuration is the config source.";
        }
        leaf running {
          type empty;
          description
            "The running configuration is the config source.";
        }
        leaf startup {
          if-feature startup;
          type empty;
          description
            "The startup configuration is the config source.";
        }
        leaf url {
          if-feature url;
          type inet:uri;
          description
            "The URL-based configuration is the config source.";
        }
        anyxml config {
          description
            "Inline Config content: 'config' element.  Represents
             an entire configuration datastore, not
             a subset of the running datastore.";
```

```
            }
          }
        }
      }
    }

    rpc delete-config {
      description
        "Delete a configuration datastore.";

      reference "RFC XXXX, section 7.4.";

      input {
        container target {
          description
            "Particular configuration to delete.";

          choice config-target {
            mandatory true;
            description
              "The configuration target to delete.";

            leaf startup {
              if-feature startup;
              type empty;
              description
                "The startup configuration is the config target.";
            }
            leaf url {
              if-feature url;
              type inet:uri;
              description
                "The URL-based configuration is the config target.";
            }
          }
        }
      }
    }

    rpc lock {
      description
        "The lock operation allows the client to lock the configuration
         system of a device.";

      reference "RFC XXXX, section 7.5.";

      input {
        container target {
```

```
            description
              "Particular configuration to lock";

            choice config-target {
              mandatory true;
              description
                "The configuration target to lock.";

              leaf candidate {
                if-feature candidate;
                type empty;
                description
                  "The candidate configuration is the config target.";
              }
              leaf running {
                type empty;
                description
                  "The running configuration is the config target.";
              }
              leaf startup {
                if-feature startup;
                type empty;
                description
                  "The startup configuration is the config target.";
              }
            }
          }
        }
      }

    rpc unlock {
      description
        "The unlock operation is used to release a configuration lock,
         previously obtained with the 'lock' operation.";

      reference "RFC XXXX, section 7.6.";

      input {
        container target {
          description
            "Particular configuration to unlock.";

          choice config-target {
            mandatory true;
            description
              "The configuration target to unlock.";

            leaf candidate {
```

```
               if-feature candidate;
               type empty;
               description
                 "The candidate configuration is the config target.";
             }
             leaf running {
               type empty;
               description
                 "The running configuration is the config target.";
             }
             leaf startup {
               if-feature startup;
               type empty;
               description
                 "The startup configuration is the config target.";
             }
           }
         }
       }
     }

     rpc get {
       description
         "Retrieve running configuration and device state information.";

       reference "RFC XXXX, section 7.7.";

       input {
         anyxml filter {
           description
             "This parameter specifies the portion of the system
              configuration and state data to retrieve.";
           nc:get-filter-element-attributes;
         }
       }

       output {
         anyxml data {
           description
             "Copy of the running datastore subset and/or state
              data which matched the filter criteria (if any).
              An empty data container indicates that the request did not
              produce any results.";
         }
       }
     }

     rpc close-session {
```

```
    description
      "Request graceful termination of a NETCONF session.";

    reference "RFC XXXX, section 7.8.";
  }

  rpc kill-session {
    description
      "Force the termination of a NETCONF session.";

    reference "RFC XXXX, section 7.9.";

    input {
      leaf session-id {
        type session-id-type;
        mandatory true;
        description
          "Particular session to kill.";
      }
    }
  }

  rpc commit {
    if-feature candidate;

    description
      "Commit the candidate configuration as the device's new
       current configuration";

    reference "RFC XXXX, section 8.3.4.1.";

    input {
      leaf confirmed {
        if-feature confirmed-commit;
        type empty;
        description
          "Requests a confirmed commit.";
        reference "RFC XXXX, section 8.3.4.1.";
      }

      leaf confirm-timeout {
        if-feature confirmed-commit;
        type uint32 {
          range "1..max";
        }
        units "seconds";
        default "600";   // 10 minutes
        description
```

```
              "The timeout interval for a confirmed commit.";
            reference "RFC XXXX, section 8.3.4.1.";
          }

        leaf persist {
          if-feature confirmed-commit;
          type string;
          description
            "This parameter is used to make a confirmed commit
             persistent.  A persistent confirmed commit is not aborted
             if the NETCONF session terminates.  The only way to abort a
             persistent confirmed commit it to let the timer expire, or
             to use the cancel-commit operation.

             The value of this parameter is a token that must be given
             in the 'persist-id' parameter of commit or cancel-commit in
             order to confirm or cancel the persistent confirmed commit.

             The token should be a random string.";
          reference "RFC XXXX, section 8.3.4.1.";
        }

        leaf persist-id {
          if-feature confirmed-commit;
          type string;
          description
            "This parameter is given in order to commit a persistent
             confirmed commit.  The value must be equal to the value
             given in the 'persist' parameter to the commit operation.
             If it does not match, the operation fails with an
             'invalid-value' error.";
          reference "RFC XXXX, section 8.3.4.1.";
        }

      }
    }

  rpc discard-changes {
    if-feature candidate;

    description
      "Revert the candidate configuration to the current
       running configuration.";

    reference "RFC XXXX, section 8.3.4.2.";
  }

  rpc cancel-commit {
```

```
      if-feature confirmed-commit;
      description
        "This operation is used to cancel an ongoing confirmed commit.
         If the confirmed commit is persistent, the parameter
         'persist-id' must be given, and it must match the value of the
         'persist' parameter.";
      reference "RFC XXXX, section 8.4.4.1.";

      input {
        leaf persist-id {
          type string;
          description
            "This parameter is given in order to cancel a persistent
             confirmed commit.  The value must be equal to the value
             given in the 'persist' parameter to the commit operation.
             If it does not match, the operation fails with an
             'invalid-value' error.";
        }
      }
    }

    rpc validate {
      if-feature validate;

      description
        "Validates the contents of the specified configuration.";

      reference "RFC XXXX, section 8.6.4.1.";

      input {
        container source {
          description
            "Particular configuration to validate.";

          choice config-source {
            mandatory true;
            description
              "The configuration source to validate.";

            leaf candidate {
              if-feature candidate;
              type empty;
              description
                "The candidate configuration is the config source.";
            }
            leaf running {
              type empty;
              description
```

```
                "The running configuration is the config source.";
          }
          leaf startup {
            if-feature startup;
            type empty;
            description
              "The startup configuration is the config source.";
          }
          leaf url {
            if-feature url;
            type inet:uri;
            description
              "The URL-based configuration is the config source.";
          }
          anyxml config {
            description
              "Inline Config content: 'config' element.  Represents
               an entire configuration datastore, not
               a subset of the running datastore.";
          }
        }
      }
    }
  }

}


  <CODE ENDS>
```

Appendix D. Capability Template

   This non-normative section defines a template that can be used to
   define protocol capabilities. Data models written in YANG usually do
   not need to define protocol capabilities since the usage of YANG
   automatically leads to a capability announcing the data model and any
   optional portions of the data model, so called features in YANG
   terminology. The capabilities template is intended to be used in
   cases where the YANG mechanisms are not powerful enough (e.g., for
   handling parametrized features) or a different data modeling language
   is used.

D.1. capability-name (template)

D.1.1. Overview

D.1.2. Dependencies

D.1.3. Capability Identifier

   The {name} capability is identified by the following capability
   string:

      {capability uri}

D.1.4. New Operations

D.1.4.1. <op-name>

D.1.5. Modifications to Existing Operations

D.1.5.1. <op-name>

   If existing operations are not modified by this capability, this
   section may be omitted.

D.1.6. Interactions with Other Capabilities

   If this capability does not interact with other capabilities, this
   section may be omitted.

Appendix E. Configuring Multiple Devices with NETCONF

   This section is non-normative.

E.1. Operations on Individual Devices

   Consider the work involved in performing a configuration update
   against a single individual device. In making a change to the
   configuration, the application needs to build trust that its change
   has been made correctly and that it has not impacted the operation of
   the device. The application (and the application user) should feel
   confident that their change has not damaged the network.

   Protecting each individual device consists of a number of steps:

   o Acquiring the configuration lock.

   o Checkpointing the running configuration.

   o Loading and validating the incoming configuration.

   o Changing the running configuration.

   o Testing the new configuration.

   o Making the change permanent (if desired).

   o Releasing the configuration lock.

   Let's look at the details of each step.

E.1.1. Acquiring the Configuration Lock

   A lock should be acquired to prevent simultaneous updates from
   multiple sources. If multiple sources are affecting the device, the
   application is hampered in both testing of its change to the
   configuration and in recovery if the update fails. Acquiring a
   short-lived lock is a simple defense to prevent other parties from
   introducing unrelated changes.

   The lock can be acquired using the <lock> operation.

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

   If the :candidate capability is supported, the candidate
   configuration should be locked.

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

E.1.2. Checkpointing the Running Configuration

   The running configuration can be saved into a local file as a
   checkpoint before loading the new configuration. If the update
   fails, the configuration can be restored by reloading the checkpoint
   file.

   The checkpoint file can be created using the <copy-config> operation.

```
      <rpc message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <copy-config>
          <target>
            <url>file://checkpoint.conf</url>
          </target>
          <source>
            <running/>
          </source>
        </copy-config>
      </rpc>
```

   To restore the checkpoint file, reverse the <source> and <target>
   parameters.

E.1.3. Loading and Validating the Incoming Configuration.

   If the :candidate capability is supported, the configuration can be
   loaded onto the device without impacting the running system.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <!-- place incoming configuration changes here -->
    </config>
  </edit-config>
</rpc>
```

   If the device supports the :validate:1.1 capability, it will by
   default validate the incoming configuration when it is loaded into
   the candidate. To avoid this validation, pass the <test-option>
   parameter with the value "set". Full validation can be requested
   with the <validate> operation.

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

E.1.4. Changing the Running Configuration

   When the incoming configuration has been safely loaded onto the
   device and validated, it is ready to impact the running system.

   If the device supports the :candidate capability, use the <commit>
   operation to set the running configuration to the candidate
   configuration. Use the <confirmed> parameter to allow automatic
   reversion to the original configuration if connectivity to the device
   fails.

```
   <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <commit>
       <confirmed/>
       <confirm-timeout>120</confirm-timeout>
     </commit>
   </rpc>
```

   If the candidate is not supported by the device, the incoming
   configuration change is loaded directly into running.

```
   <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
       <target>
         <running/>
       </target>
       <config>
         <!-- place incoming configuration changes here -->
       </config>
     </edit-config>
   </rpc>
```

E.1.5. Testing the New Configuration

   Now that the incoming configuration has been integrated into the
   running configuration, the application needs to gain trust that the
   change has affected the device in the way intended without affecting
   it negatively.

   To gain this confidence, the application can run tests of the
   operational state of the device. The nature of the test is dependent
   on the nature of the change and is outside the scope of this
   document. Such tests may include reachability from the system
   running the application (using ping), changes in reachability to the
   rest of the network (by comparing the device's routing table), or
   inspection of the particular change (looking for operational evidence
   of the BGP peer that was just added).

E.1.6. Making the Change Permanent

   When the configuration change is in place and the application has
   sufficient faith in the proper function of this change, the
   application is expected to make the change permanent.

   If the device supports the :startup capability, the current
   configuration can be saved to the startup configuration by using the
   startup configuration as the target of the <copy-config> operation.

```
   <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <copy-config>
       <target>
         <startup/>
       </target>
       <source>
         <running/>
       </source>
     </copy-config>
   </rpc>
```

   If the device supports the :candidate capability and a confirmed
   commit was requested, the confirming commit must be sent before the
   timeout expires.

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

E.1.7. Releasing the Configuration Lock

   When the configuration update is complete, the lock must be released,
   allowing other applications access to the configuration.

   Use the <unlock> operation to release the configuration lock.

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

   If the :candidate capability is supported, the candidate
   configuration should be unlocked.

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

E.2. Operations on Multiple Devices

   When a configuration change requires updates across a number of
   devices, care needs to be taken to provide the required transaction
   semantics. The NETCONF protocol contains sufficient primitives upon
   which transaction-oriented operations can be built. Providing
   complete transactional semantics across multiple devices is
   prohibitively expensive, but the size and number of windows for
   failure scenarios can be reduced.

   There are two classes of multi-device operations. The first class
   allows the operation to fail on individual devices without requiring
   all devices to revert to their original state. The operation can be
   retried at a later time, or its failure simply reported to the user.
   An example of this class might be adding an NTP server. For this
   class of operations, failure avoidance and recovery are focused on
   the individual device. This means recovery of the device, reporting
   the failure, and perhaps scheduling another attempt.

   The second class is more interesting, requiring that the operation
   should complete on all devices or be fully reversed. The network
   should either be transformed into a new state or be reset to its
   original state. For example, a change to a VPN may require updates
   to a number of devices. Another example of this might be adding a
   class-of-service definition. Leaving the network in a state where
   only a portion of the devices have been updated with the new
   definition will lead to future failures when the definition is
   referenced.

   To give transactional semantics, the same steps used in single device
   operations listed above are used, but are performed in parallel
   across all devices. Configuration locks should be acquired on all
   target devices and kept until all devices are updated and the changes
   made permanent. Configuration changes should be uploaded and
   validation performed across all devices. Checkpoints should be made
   on each device. Then the running configuration can be changed,
   tested, and made permanent. If any of these steps fail, the previous
   configurations can be restored on any devices upon which they were
   changed. After the changes have been completely implemented or
   completely discarded, the locks on each device can be released.

Appendix F. Changes from RFC 4741

   This section lists major changes between this document and RFC 4741.

   o Added the "malformed-message" error-tag.

   o Added "remove" enumeration value to the "operation" attribute.

   o Obsoleted the "partial-operation" error-tag enumeration value.

   o Added <persist> and <persist-id> parameters to the <commit>
      operation.

   o Updated the base protocol URI and clarified the <hello> message
      exchange to select and identify the base protocol version in use
      for a particular session.

   o Added a YANG module to model the operations and removed the
      operation layer from the XSD.

   o Clarified lock behavior for the candidate datastore.

   o Clarified the error response server requirements for the "delete"
      enumeration value of the "operation" attribute.

   o Added a namespace wildcarding mechanism for subtree filtering.

   o Added a "test-only" value for the <test-option> parameter to the
      <edit-config> operation.

   o Added a <cancel-commit> operation.

   o Introduced a NETCONF username and a requirement for transport
      protocols to explain how a username is derived.

Authors' Addresses

   Rob Enns (editor)
   Juniper Networks

   Email: rob.enns@gmail.com

   Martin Bjorklund (editor)
   Tail-f Systems

   Email: mbj@tail-f.com

   Juergen Schoenwaelder (editor)
   Jacobs University

   Email: j.schoenwaelder@jacobs-university.de

   Andy Bierman (editor)
   Brocade

   Email: andy.bierman@brocade.com