XMLCONF Configuration Protocol
draft-enns-xmlconf-spec-01

Status of this Memo

This document is an Internet-Draft and is in full conformance with
all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF), its areas, and its working groups. Note that
other groups may also distribute working documents as
Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://
www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 28, 2003.

Copyright Notice

Abstract

There is a need for standardized mechanisms to manipulate, install,
edit, and delete the configuration of a network device. In addition,
there is a need to retrieve device state information and receive
asynchronous device state messages in a manner consistent with the
configuration mechanisms. There is great interest in using an
XML-based data encoding because a significant set of tools for
manipulating ASCII text and XML encoded data already exists.

Table of Contents

[1](#). Introduction

The XMLCONF protocol defines a simple mechanism through which a
network device can be managed. Configuration data, state
information, and system notifications can be retrieved. 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 straight-forward API to
send and receive full and partial configuration data sets.

XMLCONF uses a remote procedure call (RPC) paradigm to define a
formal API for the network device. A client encodes an RPC in XML
[1] and sends it to a server using 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 XMLCONF is an attempt to allow the functionality of
the API 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.

XMLCONF allows a client to discover the set of protocol extensions
supported by the 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 vendor-specific capabilities can
be defined with semantic and syntactic rigor.

The XMLCONF 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. XMLCONF can be used in concert with XML-based
transformation technologies such as XSLT 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 [8] scripts from a
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 XMLCONF
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 [2].

1.1 **Protocol Overview**

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

   XMLCONF can be conceptually partitioned into four layers:

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

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

   2. The RPC layer provides a simple, transport-independent framing
       mechanism for encoding RPCs. Section 3 documents this protocol.

   3. The operations layer defines a set of base operations invoked as
       RPC methods with XML-encoded parameters. Section 5 details the
       list of base operations.

   4. The content layer is outside the scope of this document. Given
       the current proprietary nature of the configuration data being
       manipulated, the specification of this content depends on the
       device vendor. It is expected that a separate effort to specify
       a standard data definition language and standard content will be
       undertaken.

1.1.1 Capabilities

An XMLCONF capability is a set of functionality that supplements the
base XMLCONF specification. The capability is identified by a
uniform resource identifier (URI). These URIs should follow the
guidelines as described in Section 6.

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 may name one or more dependent
capabilities. These capabilities must be implemented before the
first capability can function properly. To support a capability, the
server MUST support any capabilities upon which it depends.

Section 6 defines the capabilities exchange that allows the client to
discover the server's capabilities. Section 6 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 may define standardized capabilities and vendors may
define proprietary ones. The URI MUST sufficiently distinguish the
naming authority to avoid naming collisions.

1.2 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 devices is performing configuration operations a
number of problems would arise if state data were included:

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

o A command to load the file would contain nonsensical commands such
   as commands to write read-only data.

o The configuration file would be too large.

To account for these issues, the XMLCONF protocol recognizes the

difference between configuration data and state data and provides
commands that operate on each independently. For example, the
<get-config> command retrieves configuration data only while the
<get-state> command retrieves state data.

Note that the XMLCONF protocol is concerned only with information
required to get the system software into its desired running state.
Other important persistent data such as user files and databases are
not treated as configuration data by the XMLCONF protocol.
Similarly, the collection of configuration files stored on a system
(for example, the configuration files themselves) is not itself
included in configuration data.

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

1.3 **Executive Commands**

The XMLCONF protocol provides for executive commands to perform other
functions on the system that ease the process of configuring the
system. Examples include resetting a line card, issuing ping and
traceroute commands, and debugging.

1.4 **Terminology**

1.4.1 **Configuration Session**

A configuration session is the logical connection between a network
administrator or network configuration application and a network
device. A device MUST support one or more concurrent sessions.
Global configuration attributes can be changed during any session,
and the affects are visible in all sessions. Session-specific
attributes affect only the session in which they are changed.

[2](#). **Transport Protocol Requirements**

   XMLCONF uses an RPC-based communication paradigm. A client sends a
   series of zero or more RPC request operations, which cause the server
   to respond with a corresponding series of RPC replies.

   The XMLCONF protocol can be layered on any transport 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.

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

[2.1](#) **Connection-oriented operation**

   XMLCONF is connection-oriented, requiring a persistent connection
   between peers. This connection must provide reliable, sequenced data
   delivery.

   XMLCONF connections are long-lived, persisting between protocol
   operations. This allows the client to make changes to the state of
   the connection that will persist for the lifetime of the connection.
   For example, authentication information specified for a connection
   remains in effect until the connection is closed.

   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 peer, the lock persists until either explicitly
   released or the server is informed that the connection has been
   terminated. If a connection is terminated while the client holds a
   lock, the server can perform any appropriate recovery.

[2.2](#) **Security and Privacy**

   XMLCONF connections must provide security and privacy. XMLCONF
   depends on the underlying protocol for this capability. An XMLCONF
   peer assumes that an appropriate level of security and privacy are
   provided independent of this document. For example, connections may
   be encrypted in TLS [[4](#)] (or SSH [[11](#)]), depending on the underlying
   protocol.

[2.3](#) **Authentication**

   XMLCONF connections must be authenticated. The underlying protocol
   is responsible for authentication. The peer assumes that the
   connection's authentication information has been validated by the

underlying protocol using sufficiently trustworthy mechanisms and
that the peer's entity can be trusted.

One goal of XMLCONF 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 defined by the device. For
example, a device that supports RADIUS [5] should use RADIUS to
authenticate XMLCONF sessions.

The authentication process should result in an entity whose
permissions and capabilities are known to the device. These
permissions must be enforced during the XMLCONF session. For
example, if the native user interface restricts users from changing
the network interface configuration, the user should not be able to
change this configuration data using XMLCONF.

2.4 Channels

XMLCONF requires two distinct communication channels and an optional
third channel.

One channel, called the "management channel", carries information for
managing the XMLCONF session.

A second channel, called the "operation channel", carries a series of
RPCs that constitute the real content of the XMLCONF session.

A third optional channel, called the "notification channel", carries
asynchronous notifications. This channel is established only if both
parties request it during the initial capabilities exchange. (See
Section 6 for more information.)

2.4.1 Management Channel

The XMLCONF session is considered to start when the management
channel is opened and ends when this channel is closed. If the
operation channel is open when the management channel is closed, it
should be closed immediately. Only one management channel can exist
within a particular session, although multiple sessions can be opened
simultaneously.

The management channel serves three main purposes:

o Advertise the capabilities supported by each peer.

o Manage outstanding RPCs on operation channels (that is, aborting
   them).

   o Send progress reports.

2.4.1.1 **Managing Operation Channel**

   Creation of the operation channel is transport-specific.

2.4.1.2 **Managing Outstanding RPCs**

   XML data streams by their nature prohibit unrelated data from being
   intermingled with normal content. This implies that an operation
   must be managed by an external data path to avoid intermixing the
   true content data with the management data. This is the origin of
   the requirement for multiple channels.

2.4.2 **Operation Channel**

   The operation channel is used to perform XMLCONF protocol operations
   using the <rpc> and <rpc-reply> tags. The RPC model is discussed in
   Section 3.

   Most of the XMLCONF operations are performed as RPCs over the
   operation channel.

2.4.3 **Notification Channel**

   The XMLCONF protocol allows for different notification profiles. A
   specific profile must be supported by both peers for the notification
   mechanism defined in that profile to be used. This document
   specifies a mapping to the Reliable Delivery for Syslog messages.

   Notifications are discussed in Section 6 and RFC 3195 [7].

3. RPC Model

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

3.1 Namespace

The <rpc>, <rpc-reply>, and <rpc-progress> elements are defined in the following namespace:

   http://ietf.org/xmlconf/1.0/base

3.2 <rpc> Element

The <rpc> element is used in both the management and operation channels.

The <rpc> element has a mandatory attribute "message-id", which is an arbitrary 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 use as an "message-id" attribute in any resulting <rpc-reply>, <rpc-abort-reply> or <rpc-progress> messages. For example:

```
    <rpc message-id="101" xmlns="http://ietf.org/xmlconf/1.0/base">
      <some-method>
        ...
      </some-method>
    </rpc>
```

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="102" xmlns="http://ietf.org/xmlconf/1.0/base">
    <my-own-method xmlns="http://example.net/me/1.0/my-own">
      <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="103" xmlns="http://ietf.org/xmlconf/1.0/base">
  <rock-the-house xmlns="http://example.net/house/1.0/rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>
```

The following example invokes the "rock-the-world" method with no
parameters:

```
<rpc message-id="104" xmlns="http://ietf.org/xmlconf/1.0/base">
  <rock-the-world xmlns="http://example.net/house/1.0/rock"/>
</rpc>
```

3.3 <rpc-reply> Element

The <rpc-reply> message is sent on the operations channel in response
to a <rpc> operation.

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.

The response name and response data are encoded as the contents of
the <rpc-reply> element. The name of the reply is an element
directly inside the <rpc-reply> element, and any data is encoded
inside this element.

For example:

```
<rpc-reply message-id="101" xmlns="http://ietf.org/xmlconf/1.0/base">
  <some-content>
    ...
  </some-content>
</rpc-reply>
```

3.4 <rpc-abort> Element

The <rpc-abort> element is sent on the management channel by the
sender of an <rpc> who desires to terminate an operation before it
completes. The <rpc-abort> element includes a mandatory attribute
"message-id", which is equal to the "message-id" attribute of the
<rpc> to be terminated.

The <rpc-abort> operation is encoded as an element with no

subelements or data. For example:

    <rpc-abort message-id="102" xmlns="http://ietf.org/xmlconf/1.0/base"/>

An <rpc-abort-reply> element is sent immediately on the management
channel. If the indicated <rpc-reply> is in progress on the
operations channel, it shall be terminated cleanly by closing all
open elements. An <rpc-error> element (see [Section 3.6](#)) should be
added to the <rpc-reply> indicating the operation being aborted. If
the <rpc-reply> has not yet begun, it should be sent containing an
<rpc-error> element. If multiple <rpc> requests are pending, the
<rpc-error> and <rpc-reply> messages must be sent in the proper
order.

If no pending operation matches the "message-id" attribute, then the
abort operation completes without error. The <rpc-abort> message can
be generated only for <rpc> requests that contain an "message-id"
attribute. If multiple <rpc> requests with the same "message-id"
exist, then only the request that was received first by the peer is
aborted.

3.5 <rpc-abort-reply> Element

The <rpc-abort-reply> message is sent on the management channel in
response to an <rpc-abort> operation.

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

The <rpc-abort-reply> operation is encoded as an empty element. For
example:

    <rpc-abort-reply message-id="102" xmlns="http://ietf.org/xmlconf/1.0/
base"/>

3.6 <rpc-error> Element

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

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

o tag: String identifying the error condition.

o error-code: Integer identifying the error condition.

o severity: String identifying the error severity, as determined by

      the device.

   o edit-path: Configuration data that provides the context for the
      command that caused the error. This can be the empty string if
      the command causing the error is located at the top level of the
      command hierarchy.

   o statement: Configuration or command that caused the error.

   o message: String describing the error condition.

   o action: Action taken by the device in response to this error.

   [ed: A list of standard error codes is TBD. Both protocol error and
   application error codes will be supported by <rpc-error>.]

       <rpc-error message-id="102" xmlns="http://ietf.org/xmlconf/1.0/base">
         <tag>EXAMPLE_MTU_RANGE</tag>
         <error-code>128</error-code>
         <severity>error</severity>
         <statement>mtu 21050;</statement>
         <message>MTU 21050 on Ethernet/1 is outside range 256..9192</message>
       </rpc-error>

3.7 <ok> Element

   The <ok> element is sent in <rpc-reply> messages if no error occurred
   during the processing of an <rpc> request. For example:

     <rpc-reply message-id="102" xmlns="http://ietf.org/xmlconf/1.0/base">
       <ok/>
     </rpc-reply>

3.8 <rpc-progress> Element

   Some operations might take a long time to process before an
   <rpc-reply> can be generated or might generate an <rpc-reply> that
   takes a long time to transmit. If the recipient of an <rpc>
   determines that the <rpc-reply> will not be generated and transmitted
   in less than N seconds, it can send a progress report with the
   <rpc-progress> message. The number of seconds, N, is implementation
   dependent.

   The <rpc-progress> element is sent on the management channel. It has
   a mandatory attribute "message-id", which is equal to the
   "message-id" attribute of the associated <rpc> on which progress is

being reported.

The <rpc-progress> element contains one or more of the optional
elements <percent-done>, <amount>, and <message>.

The <percent-done> element contains an estimate of the percentage of
the operation that is complete in terms of real time (i.e., wall
clock time). For example:

```
<rpc-progress message-id="103">
  <percent-done>45</percent-done>
</rpc-progress>
```

The <amount> element contains an absolute quantity indicating an
amount of work completed. For example:

```
<rpc-progress message-id="103">
   <amount>45KB</amount>
</rpc-progress>
```

The <message> element contains a text message indicating progress on
the associated <rpc>. For example:

```
<rpc-progress message-id="103">
  <message>Connecting...</message>
</rpc-progress>
<rpc-progress message-id="103">
   <message>Connected.</message>
</rpc-progress>
```

Multiple <rpc-progress> messages can be sent for a particular <rpc>.

3.9 Pipelining

The operations channel is processed serially by the managed device.
Additional <rpc> requests may be sent before previous ones have been
completed, but they are added to the queue for that channel. On any
given operations channel, the managed device may send responses only
in the order the requests were received.

Messages may be received asynchronously on the notification channel.

[4](). Configuration Model

   XMLCONF provides an initial set of operations and a number of
   capabilities that can be used to extend the base. XMLCONF peers
   exchange device capabilities when the session is initiated as
   described in [Section 6.1]().

[4.1]() Configuration Datastores

   XMLCONF 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 following configuration datastores are present in the base model.
   Capabilities may define additional configuration datastores, which
   then are available only on devices that advertise the capabilities.

   o Running: 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. XMLCONF protocol
      operations refer to this datastore using the <running> element.

[5](). **Protocol Operations**

   The XMLCONF 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-config

   o edit-config

   o copy-config

   o delete-config

   o get-state

   o kill-session

   A protocol operation may 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 XML schema in [Section 9](). The following sections
   describe the semantics of each protocol operation.

[5.1]() **<get-config>**

   Description:

        Retrieve all or part of a specified configuration.

   Parameters:

      source: @config-name

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

      config: @element-subtree

           Portions of the configuration command subtree to retrieve.
           The namespace of this configuration should be specified as

an attribute of this parameter. If this parameter is empty,
the entire configuration is returned. If the format
parameter is equal to "text", the contents of this parameter
are proprietary.

   format: (xml | text)

       Format of the return text, either "xml" or "text". If this
       parameter contains the value "xml", the contents of the
       "config" parameter are expected to conform to the XML
       Namespace specified in that parameter. If the value is
       "text", the contents of the "config" parameter are
       proprietary.

   Positive Response:

       If the device can satisfy the request, the server sends an
       <rpc-reply> element containing a <config> 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:

```
<rpc message-id="105" xmlns="http://ietf.org/xmlconf/1.0/base">
  <get-config>
    <source>
      <running/>
    </source>
    <config xmlns="http://example.com/schema/1.2/config">
      <users/>
    </config>
    <format>xml</format>
  </get-config>
</rpc>

<rpc-reply message-id="105" xmlns="http://ietf.org/xmlconf/1.0/base">
  <config xmlns="http://example.com/schema/1.2/config">
    <users>
      <user>
        <name>root</name>
        <type>superuser</type>
        <full-name>Charlie Root</full-name>
      </user>
      <user>
        <name>fred</name>
        <type>admin</type>
        <full-name>Fred Flintstone</full-name>
      </user>
      <user>
        <name>barney</name>
        <type>admin</type>
        <full-name>Barney Rubble</full-name>
      </user>
    </users>
  </config>
</rpc-reply>
```

The following example shows how additional nesting within the
<config> parameter can be used to filter more of the output in the
response:

```
       <rpc message-id="106" xmlns="http://ietf.org/xmlconf/1.0/base">
         <get-config>
           <source>
             <running/>
           </source>
           <config xmlns="http://example.com/schema/1.2/config">
             <users>
               <user>
                 <name>fred</name>
               </user>
             </users>
           </config>
           <format>xml</format>
         </get-config>
       </rpc>

       <rpc-reply message-id="106" xmlns="http://ietf.org/xmlconf/1.0/base">
         <config xmlns="http://example.com/schema/1.2/config">
           <users>
             <user>
               <name>fred</name>
               <type>admin</type>
               <full-name>Fred Flintstone</full-name>
             </user>
           </users>
         </config>
       </rpc-reply>
```

5.2 <edit-config>

   Description:

        Load all or part of a specified configuration to the specified
        target configuration. 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 does not exist, it is created.

        The device analyzes the source and target configurations and
        performs the requested changes. The target configuration is
        not simply replaced, as with the <copy-config> command.

   Attributes:

      operation: (merge | replace | delete) [default: merge]

Elements in the <config> subtree may contain an operation
attribute. The attribute identifies the point in the
configuration to perform the operation.

In the interest of simplicity, all operation attributes
appearing within the <config> subtree MUST have the same value.

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.

replace: The configuration data identified by the element
containing this attribute replaces any related configuration
in the configuration datastore identified by the target
parameter. Unlike a <copy-config> operation, which replaces
the entire target configuration, only the configuration
actually present in the config parameter is affected.

delete: The configuration data identified by the element
containing this attribute is deleted in the configuration
datastore identified by the target parameter.

[ed. The operation attribute needs to be added to the XML
schema in Section 9.]

Parameters:

target: @config-name

Configuration datastore being edited, such as <running>.

test-option: (test-then-set | set) [default: set]

test-then-set: Perform a validation test before attempting
to set; skip set if any errors.

set: Perform a set without a validation test first.

The test-option element may be specified only if the device
advertises the #validate capability (Section 6.4).

       error-option: (stop-on-error | ignore-error) [default:
       stop-on-error]

              stop-on-error: Abort the rpc request on first error.

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

       config: @element-tree

              Portion of the configuration subtree to set. The namespace
              of this configuration should be specified as an attribute of
              this parameter.

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

       <rpc message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
         <edit-config>
           <target>
             <running/>
           </target>
           <config xmlns="http://example.com/schema/1.2/config">
             <interface>
               <name>Ethernet0/0</name>
               <mtu>1500</mtu>
             </interface>
           </config>
         </edit-config>
       </rpc>

       <rpc-reply message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
         <ok/>
       </rpc-reply>

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

```
<rpc message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns="http://example.com/schema/1.2/config"
            xmlns:xc="http://ietf.org/xmlconf/1.0/base">
      <interface xc:operation="replace">
        <name>Ethernet0/0</name>
        <mtu>1500</mtu>
        <address>
          <name>1.2.3.4</name>
          <mask>255.0.0.0</mask>
        </address>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

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

```
<rpc message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns="http://example.com/schema/1.2/config"
            xmlns:xc="http://ietf.org/xmlconf/1.0/base">
      <interface xc:operation="delete">
        <name>Ethernet0/0</name>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

      Delete interface 192.168.0.1 from an OSPF area (other interfaces
      configured in the same area are unaffected):

       <rpc message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
         <edit-config>
           <target>
             <running/>
           </target>
           <config xmlns="http://example.com/schema/1.2/config"
                   xmlns:xc="http://ietf.org/xmlconf/1.0/base">
             <protocols>
               <ospf>
                 <area>
                   <name>0.0.0.0</name>
                   <interfaces>
                     <interface xc:operation="delete">
                       <name>192.168.0.1</name>
                     </interface>
                   </interfaces>
                 </area>
               </ospf>
             </protocols>
           </config>
         </edit-config>
       </rpc>

       <rpc-reply message-id="107" xmlns="http://ietf.org/xmlconf/1.0/base">
         <ok/>
       </rpc-reply>

5.3 <copy-config>

   Description:

         Create or replace an entire configuration file with the
         contents of another complete configuration file. If the target
         file exists, it is overwritten; otherwise, a new file is
         created.

         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. The source and target parameters cannot
         identify the same file.

         The device may choose not to support format conversions with
         this operation. The running and startup configurations are

considered to be format neutral, but all other configuration
files are created in a specific format (text or XML). A copy
operation on any of these format-specific files may fail if the
format parameter specifies a value different than the source
file format. It is suggested that the format parameter be
omitted in this type of operation, to select the source file
format.

Parameters:

   source: @config-name | config

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

   target: @config-name

       Name of the configuration datastore to use as the
       destination of the copy operation.

   format: (xml | text) [Default: xml]

       Format of the configuration file, either "xml" or "text".
       The format of the source and target configurations must
       match. Configuration datastores (such as <running>) match
       either format.

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="108" xmlns="http://ietf.org/xmlconf/1.0/base">
  <copy-config>
    <source>
      <running/>
    </source>
    <target>
      <url>ftp://example.com/configs/testbed-dec10.txt</url>
    </target>
    <format>text</format>
  </copy-config>
</rpc>

<rpc-reply message-id="108" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

5.4 <delete-config>

Description:

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

Parameters:

> target: @config-name
>
> > 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="108" xmlns="http://ietf.org/xmlconf/1.0/base">
          <delete-config>
            <target>
              <startup/>
            </target>
          </delete-config>
        </rpc>

        <rpc-reply message-id="108" xmlns="http://ietf.org/xmlconf/1.0/base">
          <ok/>
        </rpc-reply>
```

5.5 <get-state>

   Description:

        Retrieve device state information. Section 10 describes the
        XML schema for XMLCONF state data.

   Parameters:

      state: (@element-subtree | text)

           If the <format> parameter is equal to "xml", this parameter
           specifies the portion of the system state subtree to
           retrieve. The namespace of this configuration should be
           specified as an attribute of this parameter. If the
           <format> parameter is equal to "text", the contents of this
           parameter are proprietary. If this parameter is empty, all
           the device state information are returned.

      format: (xml | text)

           Format of the <state> parameter and the return text, either
           "xml" or "text".

   Positive Response:

        If the device was able to satisfy the request, an <rpc-reply>
        is sent. The <state> 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="109" xmlns="http://ietf.org/xmlconf/1.0/base">
  <get-state>
    <state xmlns="http://example.com/schema/1.2/int-stats">
      <interface name="ethernet0/1">
          <intstats></intstats>
      </interface>
    </state>
    <format>xml</format>
  </get-state>
</rpc>

<rpc-reply message-id="109" xmlns="http://ietf.org/xmlconf/1.0/base">
   <state xmlns="http://example.com/schema/1.2/int-stats">
      <interface name="ethernet0/1">
         <intstats>
             <inPkts>9456823</inPkts>
             <inOctets>1228484566</inOctets>
             <inErrors>4326</inErrors>
             <outPkts>4821050</outPkts>
             <outOctets>634712154</outOctets>
             <outErrors>2096</outErrors>
         </intstats>
      </interface>
   </state>
</rpc-reply>
```

[5.6](#) <kill-session>

Description:

     Force the termination of an XMLCONF session.

Parameters:

   session-id: (Positive Integer)

        Session identifier of the XMLCONF session to be terminated.
        If this value is equal to the current session ID, a 'Bad
        Value' error is sent.

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="110" xmlns="http://ietf.org/xmlconf/1.0/base">
          <kill-session>
            <session-id>4</session-id>
          </kill-session>
        </rpc>

        <rpc-reply message-id="110" xmlns="http://ietf.org/xmlconf/1.0/base">
          <ok/>
        </rpc-reply>

6. **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 be able
   to process and 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
   A. Future capability definitions may be published as standards by
   standards bodies or published as propriety by vendors.

   A capability is identified with a URI, in the form:

      http://{naming authority}/{protocol}/{version}/{category}#{name}

   Capabilities defined in this document have the following format:

      http://ietf.org/xmlconf/1.0/base#{name}

   where {name} is the name of the capability. Capabilities are often
   referenced in discussions and email using the shorthand #{name}. For
   example, the foo capability would have the formal name "http://
   ietf.org/xmlconf/1.0/base#foo" and be called "#foo". The shorthand
   form MUST NOT be used inside the protocol.

6.1 **Capabilities Exchange**

   An XMLCONF capability is a set of additional functionality
   implemented on top of the base XMLCONF specification. The capability
   is distinguished by a URI. These URIs should follow the guidelines
   as described in Section 7.7.

   Capabilities are advertised in messages sent on the management
   channel when each peer starts operation. When the management channel
   is pened, each peer sends a <hello> element containing a list of that
   peer's capabilities.

   In the following example, the peer advertises the base XMLCONF
   capability, one XMLCONF capability defined in the base XMLCONF
   document, and one vendor-specific capability.

```
<hello>
  <capabilities>
    <capability>http://ietf.org/xmlconf/1.0/base</capability>
    <capability>http://ietf.org/xmlconf/1.0/base#lock</capability>
    <capability>http:/example.net/router/2.3/core#cool-feature</capability>
  </capabilities>
</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.

6.2 Writable-Running Capability

6.2.1 Description

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

6.2.2 Dependencies

None.

6.2.3 Capability and Namespace

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

   http://ietf.org/xmlconf/1.0/base#writable-running

The #writable-running capability uses the base XMLCONF namespace URI.

6.2.4 New Operations

None.

6.2.5 Modifications to Existing Operations

6.2.5.1 <edit-config>

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

6.2.5.2 <copy-config>

The #writable-running capability modifies the <copy-config> operation

to accept the <running> element as a <target>.

6.3 Candidate Configuration Capability

6.3.1 Description

The candidate configuration capability, #candidate, indicates that
the device supports a candidate configuration datastore, which is
used to hold configuration data that can 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 a manipulating configuration data. Additions,
deletions, and changes may 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 candidate configuration can be used as a source or target of any
operation with a <source> or <target> parameter. The <candidate>
element is used to indicate the candidate configuration:

```
<rpc message-id="112" xmlns="http://ietf.org/xmlconf/1.0/base">
  <operation>
    <source>
      <candidate/>
    </source>
  </operation>
</rpc>
```

The candidate configuration may be shared among multiple sessions.
Unless a client has specific information that the candidate
configuration is not shared (for example, through another
capability), it must assume that other sessions may be 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 changes since the last <commit> operation
by executing the <discard-changes> operation. The candidate
configuration's content then reverts to the current committed
configuration.

6.3.2 Dependencies

The #candidate capability requires that the #lock capability be
implemented. Manipulation of a candidate configuration without a
locking mechanism is not advised.

6.3.3 **Capability and Namespace**

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

http://ietf.org/xmlconf/1.0/base#candidate

The #candidate capability uses the base XMLCONF namespace URI.

6.3.4 **New Operations**

6.3.4.1 **<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 system does not have the #candidate capability, the
<commit> operation is not available.

Parameters:

confirmed:

The <confirmed> element indicates that the <commit>
operation MUST be reverted if a confirming commit is not
issued within ten (10) minutes. The timeout period can be
adjusted with the <confirm-timeout> element.

confirmed-timeout: Timeout period for confirmed commit, in
minutes.

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="113" xmlns="http://ietf.org/xmlconf/1.0/base">
  <commit/>
</rpc>

<rpc-reply message-id="113" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>


<rpc message-id="114" xmlns="http://ietf.org/xmlconf/1.0/base">
  <commit>
    <confirmed/>
    <confirm-timeout>20</confirmed-timeout>
  </commit>
</rpc>

<rpc-reply message-id="114" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

6.3.4.2 <discard-changes>

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

```
<rpc xmlns="http://ietf.org/xmlconf/1.0/base">
  <discard-changes/>
</rpc>
```

This operation discards any uncommitted changes.

6.3.5 Modifications to Existing Operations

6.3.5.1 <lock> and <unlock>

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

```
<rpc message-id="115" xmlns="http://ietf.org/xmlconf/1.0/base">
  <lock>
    <source>
      <candidate/>
    </source>
  </lock>
</rpc>
```

Devices implementing the #candidate capability WILL NOT allow a
configuration lock to be acquired when there are outstanding changes
to the candidate configuration. An error WILL be returned and the
status of the lock will remain unchanged.

When a client fails with outstanding changes to the candidate
configuration, recovery can be difficult. To facilitate easy
recovery, the #candidate capability adds a <discard-changes> element
to the <lock> operation. If this element contains the value
"automatic", any outstanding changes are discarded when the lock is
released, whether explicitly with the <unlock> operation or
implicitly from session failure.

```
<rpc message-id="116" xmlns="http://ietf.org/xmlconf/1.0/base">
  <lock>
    <source>
      <candidate/>
    </source>
    <discard-changes>automatic</discard-changes>
  </lock>
</rpc>
```

6.3.5.2 <get-config> and <edit-config>

The candidate configuration is the default target for the
<edit-config> and <get-config> operations. It may be explicitly
named using the <candidate> element:

```
<rpc message-id="117" xmlns="http://ietf.org/xmlconf/1.0/base">
  <get-config>
    <source>
      <candidate/>
    </source>
  </get-config>
</rpc>
```

6.4 Validate Capability

6.4.1 Description

   Validation consists of checking a candidate 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 validate parameter to the
   <edit-config> operation and, when it is provided, checks at least for
   syntax errors.

6.4.2 Dependencies

   None.

6.4.3 Capability and Namespace

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

      http://ietf.org/xmlconf/1.0/base#validate

   The #validate capability uses the base XMLCONF namespace URI.

6.4.4 New Operations

6.4.4.1 <validate>

   Description:

        This protocol operation validates the contents of the specified
        configuration.

   Parameters:

      source: @config-name

        Name of the configuration datastore being validated, such as
        <candidate>.

   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 any of the following reasons:

+ Syntax errors

+ Missing parameters

+ References to undefined configuration data

Example:

```
<rpc message-id="118" xmlns="http://ietf.org/xmlconf/1.0/base">
  <validate>
    <candidate/>
  </validate>
</rpc>

<rpc-reply message-id="118" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

6.5 Distinct Startup Capability

6.5.1 Description

The device supports separate running and startup configuration
datastores. Operations which affect the running configuration will
not be automatically copied to the startup configuration. An
explicit <copy-config> operation from the <running> to the <startup>
must be invoked to update the startup configuration to the current
contents of the running configuration. XMLCONF protocol operations
refer to the startup datastore using the <startup> element.

6.5.2 Dependencies

None.

6.5.3 Capability and Namespace

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

http://ietf.org/xmlconf/1.0/base#startup

The #startup capability uses the base XMLCONF namespace URI.

6.5.4 New Operations

None.

6.5.5 Modifications to Existing Operations

6.5.5.1 <copy-config>

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

```
<rpc message-id="119" xmlns="http://ietf.org/xmlconf/1.0/base">
  <copy-config>
    <source>
      <running/>
    </source>
    <target>
      <startup/>
    </target>
    <format>text</format>
  </copy-config>
</rpc>
```

6.6 Lock Capability

6.6.1 Description

The #lock capability allows the client to lock the configuration
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 XMLCONF clients, non-XMLCONF clients (SNMP and Expect scripts)
and human users.

An attempt to lock the configuration MUST fail if an existing session
currently holds the lock.

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 XMLCONF
session closes. The session closure may be explicitly performed by

the client, or implicitly performed by the server based on criteria
such as lack of network connectivity, failure of the underlying
transport, or simple inactivity timeout. This criteria is dependent
on the vendor's implementation and the underlying transport.

The lock operation takes an optional parameter, target. If the
target parameter is specified, it names the configuration that will
be locked. If the target parameter is not specified, then all
configurations will be locked. When a lock is active, <edit-config>
and <copy-config> operations will be disallowed on the locked
configuration(s) by any other session. Additionally, the system will
ensure that these locked configuration resources will not be modified
by other non-XMLCONF management operations such as SNMP and CLI. The
<kill-session> command (at the RPC layer) can be used to force the
release of a lock.

6.6.2 Dependencies

None.

6.6.3 Capability and Namespace

The #lock capability is identified by the following capability
string:

http://ietf.org/xmlconf/1.0/base#lock

The #lock capability uses the base XMLCONF namespace URI.

6.6.4 New Operations

6.6.4.1 <lock>

Description:

A configuration source can be locked by using the <lock>
operation. A lock will not be granted if any of the following
conditions are true:

+ a lock is already held by another session

+ the target configuration has already been modified and these
   changes have not been committed

+ lock capability not supported

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: @config-name [Optional]

Name of the configuration datastore to lock. If this
parameter is not present, than all configuration datastores
will be locked.

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. This error
response will include the session number of the lock owner (if
failure due to lock already held).

Example:

```
<rpc message-id="120" xmlns="http://ietf.org/xmlconf/1.0/base">
  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>

<rpc-reply message-id="120" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

6.6.4.2 **<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
           <error>.

     Parameters:

        target: @config-name [Optional]

               Name of the configuration datastore to unlock. If this
               parameter is not present, than all configuration datastores
               will be unlocked.

               An XMLCONF 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="121" xmlns="http://ietf.org/xmlconf/1.0/base">
          <unlock>
            <target>
             <running/>
            </target>
          </unlock>
        </rpc>

        <rpc-reply message-id="121" xmlns="http://ietf.org/xmlconf/1.0/base">
          <ok/>
        </rpc-reply>

6.7 Notifications Capability

6.7.1 **Description**

   The #notifications capability indicates that the server supports the
   notification channel. This channel provides a mechanism for sending
   asynchronous notifications within the XMLCONF session. This channel
   can be used for events and system logging.

6.7.2 **Dependencies**

   None.

6.7.3 **Capability and Namespace**

   The #notifications capability is identified by the following
   capability string:

      http://ietf.org/xmlconf/1.0/base#notifications

   The #notifications capability uses the base XMLCONF namespace URI.

6.7.4 **New Operations**

6.7.4.1 **<open-notifications>**

   Description:

        Use the <open-notifications> operation to request the
        notification channel with a specific set of parameters. If
        successful, the underlying protocol will open the notification
        channel with the appropriate parameters.

   Parameters:

     format: format

          Indicates the format of the notification channel. The only
          legal value is "rfc3195".

     matching: match-expression [Optional]

          An optional parameter that limits notifications sent on the
          channel to those matching the match-expression.

   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="122" xmlns="http://ietf.org/xmlconf/1.0/base">
  <open-notifications>
    <format>rfc3195</format>
    <matching>match-expression</matching>
  </open-notifications>
</rpc>

<rpc-reply message-id="122" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

6.7.4.2 <close-notifications>

Description:

> Use the <close-notifications> operation to close the
> notification channel. If successful, the underlying protocol
> will close the notification channel.

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="123" xmlns="http://ietf.org/xmlconf/1.0/base">
  <close-notifications/>
</rpc>

<rpc-reply message-id="123" xmlns="http://ietf.org/xmlconf/1.0/base">
  <ok/>
</rpc-reply>
```

6.8 **URL Capability**

6.8.1 **Description**

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

6.8.2 **Dependencies**

   None.

6.8.3 **Capability and Namespace**

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

     http://ietf.org/xmlconf/1.0/base#url?protocol={protocol-name,...}

   The #url capability uses the base XMLCONF namespace URI.

   The #url capability URI MUST contain a "protocol" argument assigned a
   comma-separated list of protocol names indicating which protocols the
   XMLCONF peer supports. For example:

     http://ietf.org/xmlconf/1.0/base#url?protocol=http,ftp,file

   The #url capability uses the base XMLCONF namespace URI.

6.8.4 **New Operations**

   None.

6.8.5 **Modifications to Existing Operations**

6.8.5.1 **<edit-config>**

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

6.8.5.2 **<copy-config>**

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

6.8.5.3 **<delete-config>**

   The #url capability modifies the <delete-config> operation to accept

the <url> element as the value of the the <target> parameters. If
this parameter contains a URL, then it should identify a local
configuration file.

6.8.5.4 <validate>

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

7. **XML Usage Guidelines for XMLCONF**

   XML serves as an encoding format for XMLCONF, 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.

   To simplify manipulation of XMLCONF content, use of XML is restricted
   to a simple subset of XML, as described in this section.

7.1 **No DTDs**

   Document type declarations (DTDs) are not permitted to appear in
   XMLCONF content.

7.2 **Avoid Mixed Content**

   Mixed content is defined as elements that can contain both data and
   other elements. Elements in XMLCONF can contain either data or
   additional elements only.

   This greatly simplifies the complexity of parsing XML, especially in
   the area of significant whitespace. Whitespace inside data elements
   is significant. Whitespace outside data elements is not.

```
<valid>
  <element>data</element>
  <more>data</more>
</valid>

<not-valid>
  <element>data<more>data</more>maybe some</element>
</not-valid>
```

7.3 **No Attributes in the Default Namespace**

   Do not use attributes in the default namespace. All attributes
   should be qualified.

   Unqualified attributes belong to the default namespace, and their use
   pollutes the default namespace. Restricting them to the current
   namespace encourages meaningful definitions that are free of
   collisions.

```
<valid xmlns="http://valid/" xmlns:v="http://valid/" v:foo="cool"></valid>

<not-valid xmlns="http://not-valid/" foo="not-cool"></not-valid>
```

7.4 **Use Container Elements for Lists**

   When encoding lists with multiple instances, use a distinct container
   element, preferable the plural form of the instance element.

   In this example, the element 'grommet' is contained within the
   'grommets' element.

      <valid>
        <grommets>
          <grommet>....</grommet>
          <grommet>....</grommet>
          <grommet>....</grommet>
        </grommets>
      </valid>

   Use of container elements allows simpler manipulation of lists and
   list members.

7.5 **Elements and Attributes**

   The choice of elements and attributes has been widely discussed, but
   no absolute guidelines exist. When designing encoding rules for
   XMLCONF content, the following guidelines should be used:

7.5.1 **Consider Attributes as Metadata**

   Attributes should contain metadata about the element, not true data.
   By extension, vital information should not be encoded in attributes.

7.5.2 **Consider the Lack of Extensibility of Attributes**

   Attributes are unordered, can appear only once, and can have no
   children. Data scenarios which must leave room for future expansion
   (in future specifications or future software releases) should avoid
   attributes.

7.6 **Proper Tag Names**

   When choosing element names, consider the following guidelines:

   o Prefer ASCII (7-bit).

   o Prefer lowercase.

   o Prefer dashes to underscores.

   o Prefer full words. Note that "config" is considered a full word.

These are guidelines only and should be considered secondary to the
need for consistency with existing vocabularies. For example, when
encoding MIB variables names in XMLCONF, use the existing names
(ifAddr) instead of shifting to these guidelines (if-address). These
guidelines are valuable when no common vocabulary exists, because
they help to avoid the scenario in which a dozen developers choose a
dozen names that differ in ways that lead to frustrating
inconsistencies, such as ifaddr, if-addr, if-address,
interface-address, intf-addr, iaddr, and iface-addr.

7.7 Namespaces

A namespace URI uniquely identifies the content and meaning of an XML
element. When designing XML namespaces for XMLCONF content, the
following guidelines should be used:

o   Prefer domain names in URIs. Use the domain name of the
    organization that controls the content of the scheme.

o   Prefer version numbers in namespaces. Use dates when version
    numbers are not appropriate. Versions should be formatted in
    strings that are consistent with the software being referenced.
    Dates should be formatted as "YYYY-MM-DD".

o   Prefer URLish URIs, but do not expect them all to be reachable or
    meaningful. While URIs are not URLs and are not required to
    reference any resource, using non-URL syntax is needlessly
    confusing. For example, the following URI looks like a programmer
    mistake:

        ietf.org:/rfc/rfc1234.txt

The model namespace looks like:

    http://${naming-authority}/${topic}/${version}/${area}

For example:

    http://ietf.org/xmlconf/1.0/base-config

In this usage, 'topic' might be the product name, 'version' might be
the software version, and 'area' might be the portion of that
software documented in this particular namespace.

    http://example.net/magic-os/84.1.3/bgp

The ${topic} segment might contain a qualifying hierarchy. For
example, if the Puff Router Company has a large set of operating

systems targeted at differing market segments, it may express this
relationship in the ${topic}:

    http://example.net/embedded/magic-os/84.1.3/bgp

8. **BEEP Mapping**

   All XMLCONF implementations MUST implement the profile and functional
   mapping between XMLCONF and BEEP as described below.

8.1 XMLCONF Session Initiation

   Managers may be either BEEP listeners or initiators. Similarly,
   agents may be either listeners or initiators. Thus the initial
   exchange takes place without regard to whether a manager or the agent
   is the initiator. After the transport connection is established, as
   greetings are exchanged, they should each announce their support for
   TLS [4] and optionally SASL [3] (see below), as well as for the
   SYSLOG profile [7]. Once greetings are exchanged, if TLS is to be
   used and available by both parties, the listener STARTs a channel
   with the TLS profile.

   Once TLS has been started, a new greeting is sent by both initiator
   and listener, as required by the BEEP RFC.

   At this point, if SASL is desired, the initiator starts BEEP channel
   1 to perform a SASL exchange to authenticate itself. When SASL is
   completed, the channel MUST be closed.

   Once authentication has occurred, there is no need to distinguish
   between initiator and listener. We now distinguish between manager
   and agent.

   The manager now establishes an XMLCONF management channel for the
   purpose of exchanging capabilities, monitoring progress, and aborting
   remote procedure calls. As initiators assign odd channels and
   listeners assign even channels, the management channel is BEEP
   channel 1 or 2, depending on whether the manager is the initiator or
   the listener.

   The manager next establishes the XMLCONF operational channel for the
   purpose of issuing RPC requests. This channel is BEEP channel 3 or
   4.

   Finally, if either manager or agent wishes to send or receive
   notifications, it may issue a start on the next available channel if
   the other side has sent the send or receive XMLCONF capability.

   At this point, the XMLCONF session is established.

8.2 XMLCONF RPC Execution

   To issue an RPC, the manager transmits on the operational channel a

BEEP MSG containing the RPC and its arguments. In accordance with
the BEEP standard, RPC requests may be split across multiple BEEP
frames.

Once received and processed, the agent responds with BEEP RPYs on the
same channel with the response to the RPC. In accordance with the
BEEP standard, responses may be split across multiple BEEP frames.

8.3 XMLCONF <rpc-abort> and <rpc-progress>

<rpc-abort> and <rpc-progress> requests are issued by the manager on
the XMLCONF management channel, and the agent responds with BEEP RPYs
on that same channel.

8.4 XMLCONF Session Teardown

Either side may initiate the termination of an XMLCONF session. In
This is done by issuing a BEEP close on the operational channel after
the current RPC has completed. The same is done with any
notification channels by the end that transmits notifications.
Finally, BEEP channel 0 is closed.

8.5 BEEP Profiles for XMLCONF Channels

There are two profiles, the management channel profile and the
operations channel profile. These are not to be confused with the
BEEP control channel.

The operations channel will have two commands, <rpc> and <rpc-reply>.
The management channel will have one additional operation with
<rpc-progress>.

8.5.1 Management Channel Profile

```
<!-- DTD for xmlconf management over BEEP

   Refer to this DTD as:

     <!ENTITY % XMLCONF PUBLIC "xmlconf/management/1.0" "">
     %XMLCONF;
   -->

<!--   Contents

     Overview

     Includes
     Profile Summaries
```

       Entity Definitions

       Operations
        rpc
        rpc-reply
        rpc-progress
     -->

     <!-- Overview   XMLCONF Management channel -->

     <!-- Includes -->

          <!ENTITY % BEEP PUBLIC "-//Blocks//DTD BEEP//EN"
                    "">
          %BEEP;

     <!-- Profile summaries

        BEEP profile XMLCONF-MANAGEMENT

        role   MSG   RPY   ERR
        ====   ===   ===   ===
        I or L   rpc   ok   error
        I or L   rpc-reply   ok   error
        I or L   rpc-progress   ok   error

     -->

     <!--
       Entity Definitions

            entity   syntax/reference   example
            ======   ================   =======

       a PRC
         RPC-DATA   Alpha
       a RPC reply number
            RPC-REPLY   1*3DIGIT
       a RPC progress number
            RPC-PROGRESS   1*3DIGIT

     -->

     <!ENTITY % RPC-REPLY   "CDATA">
     <!ENTITY % RPC-DATA   "CDATA">
     <!ENTITY % RPC-PROGRESS "CDATA">
      -->

```
     <!--
       RPC command
       -->

     <!ELEMENT rpc          (#ELEMENTDATA)>
     <!ATTLIST rpc
             rpc-data       %RPC_DATA;                   #REQUIRED>


   <!--
       Result of RPC.
       -->

     <!ELEMENT rpc-reply    (#ELEMENTDATA)>
     <!ATTLIST rpc-reply
             rpc-reply      %RPC-REPLY;                  #REQUIRED
         rpc-data     %rpc-data                  #REQUIRED>

   <!--
       Progress of RPC operation.
       -->

     <!ELEMENT rpc-progress   (#ELEMENTDATA)>
     <!ATTLIST rpc-progress
             rpc-progress %RPC-PROGRESS;                 #REQUIRED>

     <!-- End of DTD -->
```

[8.5.2](#) Operations Channel Profile

```
     <!-- DTD for xmlconf operations over BEEP

       Refer to this DTD as:

         <!ENTITY % XMLCONF PUBLIC "xmlconf/Operation/1.0" "">
         %XMLCONF;
       -->

     <!--   Contents

         Overview

         Includes
         Profile Summaries
         Entity Definitions

         Operations
```

```
         rpc
         rpc-reply
      -->


      <!--  Overview   XMLCONF operation channel  -->


      <!-- Includes -->


           <!ENTITY % BEEP PUBLIC "-//Blocks//DTD BEEP//EN"
                      "">
           %BEEP;

      <!--  Profile summaries


         BEEP profile XMLCONF-MANAGEMENT

         role         MSG                       RPY         ERR
         ====         ===                       ===         ===
         I or L       rpc                                   ok          error
         I or L       rpc-reply                 ok          error


      -->


      <!--
        Entity Definitions

              entity          syntax/reference      example
              ======          ================      =======

        a PRC
           RPC-DATA         Alpha
        a RPC reply number
              RPC-REPLY        1*3DIGIT


      -->

      <!ENTITY % RPC-REPLY     "CDATA">
      <!ENTITY % RPC-DATA      "CDATA">

       -->


      <!--
        RPC command
        -->

      <!ELEMENT RPC         (#ELEMENTDATA)>
      <!ATTLIST RPC
```

```
          RPC-DATA        %RPC_DATA;                    #REQUIRED>


  <!--
      Result of RPC.
      -->

      <!ELEMENT RPC-REPLY    (#ELEMENTDATA)>
      <!ATTLIST RPC-REPLY
              RPC-REPLY      %RPC-REPLY;                  #REQUIRED
              RPC-DATA    %RPC-DATA            #REQUIRED>


      <!-- End of DTD -->
```

8.5.3 **Notification Channel Profile**

The XMLCONF notification channel profile is defined in RFC 3195 [7].

9. **XML Schema for XMLCONF RPC and Protocol Operations**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://ietf.org/xmlconf/1.0/base"
            xmlns:xc="http://ietf.org/xmlconf/1.0/base"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="unqualified">
  <xsd:complexType name="rpcType">
    <xsd:sequence>
      <xsd:element ref="xc:rpcOperation"/>
    </xsd:sequence>
    <xsd:attribute name="message-id" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="rpc" type="xc:rpcType"/>
  <xsd:complexType name="rpc-errorType">
    <xsd:sequence>
      <xsd:element name="tag" type="xsd:string" minOccurs="0"/>
      <xsd:element name="error-code" type="xsd:integer" minOccurs="0"/>
      <xsd:element name="severity" type="xsd:string" minOccurs="0"/>
      <xsd:element name="edit-path" type="xsd:string" minOccurs="0"/>
      <xsd:element name="statement" type="xsd:string" minOccurs="0"/>
      <xsd:element name="message" type="xsd:string" minOccurs="0"/>
      <xsd:element name="action" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="rpc-replyType">
    <xsd:choice>
      <xsd:element name="ok" minOccurs="0"/>
      <xsd:element name="rpc-error"
                   type="xc:rpc-errorType" minOccurs="0"/>
      <xsd:element ref="xc:config" minOccurs="0"/>
      <xsd:element ref="xc:state" minOccurs="0"/>
    </xsd:choice>
    <xsd:attribute name="message-id" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="rpc-reply" type="xc:rpc-replyType"/>
  <xsd:element name="percent-done" type="xsd:string"/>
  <xsd:element name="amount" type="xsd:string"/>
  <xsd:element name="message" type="xsd:string"/>
  <xsd:complexType name="rpc-progressType">
    <xsd:choice>
      <xsd:element ref="xc:percent-done"/>
      <xsd:element ref="xc:amount"/>
      <xsd:element ref="xc:message"/>
    </xsd:choice>
    <xsd:attribute name="message-id" type="xsd:string" use="required"/>
  </xsd:complexType>
```

```
    <xsd:element name="rpc-progress" type="xc:rpc-progressType"/>
    <xsd:complexType name="rpc-abortType">
      <xsd:attribute name="message-id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="rpc-abort" type="xc:rpc-abortType"/>
    <xsd:complexType name="rpc-abort-replyType">
      <xsd:attribute name="message-id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="rpc-abort-reply" type="xc:rpc-abort-replyType"/>
    <xsd:simpleType name="responseAttributeType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="terse"/>
        <xsd:enumeration value="full"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="test-optionType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="test"/>
        <xsd:enumeration value="test-then-set"/>
        <xsd:enumeration value="set"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="test-option" type="xc:test-optionType"/>
    <xsd:simpleType name="error-optionType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="stop-on-error"/>
        <xsd:enumeration value="ignore-error"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="error-option" type="xc:error-optionType"/>
    <xsd:complexType name="rpcOperationType">
      <xsd:attribute name="response" type="xc:responseAttributeType"
                     default="terse"/>
    </xsd:complexType>
    <xsd:element name="rpcOperation" type="xc:rpcOperationType"
                 abstract="true"/>
    <xsd:simpleType name="configFormatType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="xml"/>
        <xsd:enumeration value="text"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="format" type="xc:configFormatType"/>
    <xsd:complexType name="config-inlineType">
      <xsd:complexContent>
        <xsd:extension base="xsd:anyType">
          <xsd:attribute name="format" type="xc:configFormatType"/>
        </xsd:extension>
```

```
          </xsd:complexContent>
        </xsd:complexType>
        <xsd:element name="config" type="xc:config-inlineType"/>
        <xsd:element name="state" type="xc:config-inlineType"/>
        <xsd:complexType name="config-nameType"/>
        <xsd:element name="config-name" type="xc:config-nameType"
                   abstract="true"/>
        <xsd:element name="startup" type="xc:config-nameType"
                 substitutionGroup="xc:config-name"/>
        <xsd:element name="candidate" type="xc:config-nameType"
                 substitutionGroup="xc:config-name"/>
        <xsd:element name="running" type="xc:config-nameType"
                 substitutionGroup="xc:config-name"/>
        <xsd:complexType name="config-uriType">
          <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
              <xsd:attribute name="format" type="xc:configFormatType"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
        <xsd:element name="url" type="xc:config-uriType"/>
        <xsd:complexType name="rpcOperationSourceType">
          <xsd:choice>
            <xsd:element ref="xc:config"/>
            <xsd:element ref="xc:config-name"/>
            <xsd:element ref="xc:url"/>
          </xsd:choice>
        </xsd:complexType>
        <xsd:element name="source" type="xc:rpcOperationSourceType"/>
        <xsd:complexType name="rpcOperationTargetType">
          <xsd:choice>
            <xsd:element ref="xc:config-name"/>
            <xsd:element ref="xc:url"/>
          </xsd:choice>
        </xsd:complexType>
        <xsd:element name="target" type="xc:rpcOperationTargetType"/>
        <xsd:complexType name="get-configType">
          <xsd:complexContent>
            <xsd:extension base="xc:rpcOperationType">
              <xsd:sequence>
                <xsd:element ref="xc:source"/>
                <xsd:element ref="xc:config" minOccurs="0"/>
                <xsd:element ref="xc:format" minOccurs="0"/>
              </xsd:sequence>
            </xsd:extension>
          </xsd:complexContent>
        </xsd:complexType>
        <xsd:element name="get-config" type="xc:get-configType"
```

```
          substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="edit-configType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:source" minOccurs="0"/>
            <xsd:element ref="xc:target"/>
            <xsd:element ref="xc:test-option" minOccurs="0"/>
            <xsd:element ref="xc:write-option" minOccurs="0"/>
            <xsd:element ref="xc:error-option" minOccurs="0"/>
            <xsd:element ref="xc:config" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="edit-config" type="xc:edit-configType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="copy-configType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:source"/>
            <xsd:element ref="xc:target"/>
            <xsd:element ref="xc:format"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="copy-config" type="xc:copy-configType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="delete-configType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:target"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="delete-config" type="xc:delete-configType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="get-stateType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:state"/>
            <xsd:element ref="xc:format" minOccurs="0"/>
          </xsd:sequence>
```

```
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="get-state" type="xc:get-stateType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="lockType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:target"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="lock" type="xc:lockType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="unlockType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:target"/>
            <xsd:element name="discard-changes" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="unlock" type="xc:unlockType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="validateType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element ref="xc:source"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="validate" type="xc:validateType"
            substitutionGroup="xc:rpcOperation"/>
    <xsd:complexType name="commitType">
      <xsd:complexContent>
        <xsd:extension base="xc:rpcOperationType">
          <xsd:sequence>
            <xsd:element name="confirmed" minOccurs="0"/>
            <xsd:element name="confirmed-timeout" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
```

```
       </xsd:complexType>
       <xsd:element name="commit" type="xc:commitType"
                 substitutionGroup="xc:rpcOperation"/>
       <xsd:complexType name="discard-changesType">
         <xsd:complexContent>
           <xsd:extension base="xc:rpcOperationType"/>
         </xsd:complexContent>
       </xsd:complexType>
       <xsd:element name="discard-changes" type="xc:discard-changesType"
                 substitutionGroup="xc:rpcOperation"/>
       <xsd:complexType name="kill-sessionType">
         <xsd:complexContent>
           <xsd:extension base="xc:rpcOperationType">
             <xsd:sequence>
               <xsd:element name="session-id" type="xsd:string" minOccurs="0"/>
             </xsd:sequence>
           </xsd:extension>
         </xsd:complexContent>
       </xsd:complexType>
       <xsd:element name="kill-session" type="xc:kill-sessionType"
                 substitutionGroup="xc:rpcOperation"/>
     </xsd:schema>
```

10. XML Schema for XMLCONF State Data


```
<schema
    targetNamespace="http://ietf.org/xmlconf/1.0/state"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xc="http://ietf.org/xmlconf/1.0/state"
    elementFormDefault="unqualified">

 <annotation>
  <documentation xml:lang="en">
   Initial schema for XMLCONF state information.
  </documentation>
 </annotation>

 <element name="xmlconf-state">
  <complexType>
   <sequence>

    <element name="capabilities">
     <annotation>
      <documentation xml:lang="en">
       List of XMLCONF capabilities supported by this device.
      </documentation>
     </annotation>
     <complexType>
      <sequence>
       <element name="capability" type="anyURI"
        minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
     </complexType>
    </element>

    <element name="sessions">
     <annotation>
      <documentation xml:lang="en">
       List of XMLCONF sessions currently active on this device.
      </documentation>
     </annotation>
     <complexType>
      <sequence>
       <element name="my-session-id" type="positiveInteger"/>
       <element name="session" type="xc:XmlconfSessionInfo"
        minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
     </complexType>
    </element>
```

```
     <element name="configs">
      <annotation>
       <documentation xml:lang="en">
        List of XMLCONF configuration databases supported on this device.
       </documentation>
      </annotation>
      <complexType>
       <sequence>
        <element name="config" type="xc:XmlconfConfigInfo"
         minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
      </complexType>
     </element>

    </sequence>
   </complexType>
  </element>

  <complexType name="XmlconfSessionInfo">
   <sequence>
    <element name="session-id" type="positiveInteger"/>
    <element name="username" type="string"/>
    <element name="login-time" type="dateTime"/>
   </sequence>
  </complexType>

  <complexType name="XmlconfConfigInfo">
   <sequence>
    <element name="config-name" type="xc:ConfigName"/>
    <element name="lock-status" type="xc:LockStatus"/>
   </sequence>
  </complexType>

  <complexType name="ConfigName">
   <choice>
    <element name="candidate"/>
    <element name="running"/>
    <element name="startup"/>
   </choice>
  </complexType>

  <complexType name="LockStatus">
   <sequence>
    <element name="lock-state">
     <simpleType>
      <restriction base="string">
       <enumeration value="locked"/>
       <enumeration value="unlocked"/>
```

```
          </restriction>
         </simpleType>
        </element>
        <element name="locked-by" type="positiveInteger"
          minOccurs="0"/>
       </sequence>
      </complexType>

     </schema>
```

[11]. Security Considerations

   Configuration information is by its very nature sensitive. Its
   transmission in the clear and without integrity checking leaves
   devices open to classic so-called "person in the middle" attacks.
   Configuration information often times contains passwords, user names,
   service descriptions, and topological information, all of which are
   sensitive.

   The protocol, therefore, must minimally support options for both
   confidentiality and authentication. The initial transport mapping
   makes use of BEEP. BEEP itself makes use of both transport layer
   security and SASL. We require that TLS be used in BEEP as described
   by the BEEP standard. Client-side certificates are strongly
   desirable, but an SASL authentication is the bare minimum. SASL
   allows for the use of protocols such as radius, so that
   authentication can occur off the box.

   SASL authentication will occur on the first channel creation. No
   further authentication may occur during the same session. This
   avoids a situation where rights are different between different
   channels. If an implementation wishes to support multiple accesses
   by different individuals with different rights, then multiple
   sessions are required.

   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 then a simple "permission denied" is sufficient. Note
   that authorization information may be exchanged in the form of
   configuration information, which is all the more reason to ensure the
   security of the connection.

[12](#). Authors and Acknowledgements

   This document was written by:

      Andy Bierman, Cisco Systems

      Ken Crozier, Cisco Systems

      Rob Enns, Juniper Networks

      Ted Goddard, Wind River

      Eliot Lear, Cisco Systems

      David Perkins, Riverstone Networks

      Phil Shafer, Juniper Networks

      Steve Waldbusser

      Margaret Wasserman, Wind River

Normative References

   [1]   Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler,
         "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C REC
         REC-xml-20001006, October 2000.

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

   [3]   Myers, J., "Simple Authentication and Security Layer (SASL)",
         RFC 2222, October 1997.

   [4]   Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and
         P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January
         1999.

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

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

   [7]   New, D. and M. Rose, "Reliable Delivery for syslog", RFC 3195,
         November 2001.

Informative References

    [8]   Clark, J., "XSL Transformations (XSLT) Version 1.0", W3C REC
          REC-xslt-19991116, November 1999.

    [9]   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.

    [10] Boyer, J., "Canonical XML Version 1.0", RFC 3076, March 2001.

    [11] Rinne, T., Ylonen, T., Kivinen, T. and S. Lehtinen, "SSH
          Protocol Architecture", draft-ietf-secsh-architecture-13 (work
          in progress), September 2002.

Author's Address

    Rob Enns
    Juniper Networks
    1194 North Mathilda Ave
    Sunnyvale, CA 94089
    US

    EMail: rpe@juniper.net

Appendix A. Capability Template

A.1 capability-name (template)

A.1.1 Overview

A.1.2 Dependencies

A.1.3 Capability and Namespace

   The {name} is identified by following capability string:

      http://ietf.org/xmlconf/1.0/base#{name}

   The {name} capability uses the base XMLCONF namespace URI.

A.1.4 New Operations

A.1.4.1 <op-name>

A.1.5 Modifications to Existing Operations

A.1.5.1 <op-name>

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

A.1.6 Interactions with Other Capabilities

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

Appendix B. Configuring Multiple Devices with XMLCONF

B.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 Loading the update.

   o Validating the incoming configuration.

   o Checkpointing the running 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.

B.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 should the update fail. Acquiring a
   short-lived lock is a simple defense to prevent other parties from
   introducing unrelated changes while.

   The lock can be acquired only if the device supports the #lock
   capability. The lock can be acquired using the <lock> operation.

```
<rpc message-id="201" xmlns="http://ietf.org/xmlconf/1.0/base">
  <lock>
    <source>
      <running/>
    </source>
  </lock>
</rpc>
```

If the #candidate capability is also supported, failure recovery can
be simplified by using the <discard-changes> parameter.

```
<rpc message-id="202" xmlns="http://ietf.org/xmlconf/1.0/base">
  <lock>
    <discard-changes>automatic</discard-changes>
    <source>
      <candidate/>
    </source>
  </lock>
</rpc>
```

B.1.2 Loading the Update

The configuration can be loaded onto the device without impacting the
running system. If the #url capability is supported, incoming
changes can be placed in a local file.

```
<rpc message-id="203" xmlns="http://ietf.org/xmlconf/1.0/base">
  <copy-config>
    <source>
      <config>
        <!-- place incoming configuration here -->
      </config>
    </source>
    <target>
      <url>file://incoming.conf</url>
    </target>
    <format>text</format>
  </copy-config>
</rpc>
```

If the #candidate capability is supported, the candidate
configuration can be used.

```
  <rpc message-id="204" xmlns="http://ietf.org/xmlconf/1.0/base">
    <edit-config>
      <source>
        <config>
          <!-- place incoming configuration here -->
        </config>
      </source>
      <target>
        <candidate/>
      </target>
    </edit-config>
  </rpc>
```

If the update fails, the user file can be deleted using the
<delete-config> operation or the candidate configuration reverted
using the <discard-changes> operation.

B.1.3 Validating the Incoming Configuration

Before applying the incoming configuration, it is often useful to
validate it. Validation allows the application to gain confidence
that the change will succeed and simplifies recovery if it does not.

If the device supports the #url capability, use the <validate>
operation with the <source> parameter set to the proper user file:

```
  <rpc message-id="205" xmlns="http://ietf.org/xmlconf/1.0/base">
    <validate>
      <source>
        <url>file://incoming.conf</url>
      </source>
    </validate>
  </rpc>
```

If the device supports the #candidate capability, some validation
will be performed as part of loading the incoming configuration into
the candidate. For full validation, either pass the <validate>
parameter during the <edit-config> step given above, or use the
<validate> operation with the <source> parameter set to <candidate>.

```
  <rpc message-id="206" xmlns="http://ietf.org/xmlconf/1.0/base">
    <validate>
      <source>
        <candidate/>
      </source>
    </validate>
  </rpc>
```

B.1.4 **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="207" xmlns="http://ietf.org/xmlconf/1.0/base">
  <copy-config>
    <source>
      <running/>
    </source>
    <target>
      <url>file://checkpoint.conf</url>
    </target>
    <format>text</format>
  </copy-config>
</rpc>
```

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

B.1.5 **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 #url capability, use the <edit-config>
   operation to merge the incoming configuration into the running
   configuration.

```
<rpc message-id="208" xmlns="http://ietf.org/xmlconf/1.0/base">
  <edit-config>
    <source>
      <url>file://incoming.conf</url>
    </source>
    <target>
      <running/>
    </target>
  </edit-config>
</rpc>
```

   If the device supports the #candidate capability, use the <commit>
   operation to set the running configuration to the candidate
   configuration. Use the <confirm> parameter to allow automatic
   reverting to the original configuration if connectivity to the device

   fails.

```
   <rpc message-id="209" xmlns="http://ietf.org/xmlconf/1.0/base">
     <commit>
       <confirmed/>
       <confirm-timeout>15</confirm-timeout>
     </commit>
   </rpc>
```

B.1.6 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).

B.1.7 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 should 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="210" xmlns="http://ietf.org/xmlconf/1.0/base">
     <copy-config>
       <source>
         <running/>
       </source>
       <target>
         <startup/>
       </target>
       <format>text</format>
     </copy-config>
   </rpc>
```

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

```
<rpc message-id="211" xmlns="http://ietf.org/xmlconf/1.0/base">
  <commit/>
</rpc>
```

B.1.8 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="212" xmlns="http://ietf.org/xmlconf/1.0/base">
  <unlock/>
</rpc>
```

B.2 Operations on Multiple Devices

When a configuration change requires updates across a number of
devices, care should be taken to provide the required transaction
semantics. The XMLCONF 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 multidevice operations. The first class of
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.
A 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 it was
changed. After the changes have been completely implemented or
completely discarded, the locks on each device can be released.

Intellectual Property Statement

   The IETF takes no position regarding the validity or scope of any
   intellectual property or other rights that might be claimed to
   pertain to the implementation or use of the technology described in
   this document or the extent to which any license under such rights
   might or might not be available; neither does it represent that it
   has made any effort to identify any such rights. Information on the
   IETF's procedures with respect to rights in standards-track and
   standards-related documentation can be found in BCP-11. Copies of
   claims of rights made available for publication and any assurances of
   licenses to be made available, or the result of an attempt made to
   obtain a general license or permission for the use of such
   proprietary rights by implementors or users of this specification can
   be obtained from the IETF Secretariat.

   The IETF invites any interested party to bring to its attention any
   copyrights, patents or patent applications, or other proprietary
   rights which may cover technology that may be required to practice
   this standard. Please address the information to the IETF Executive
   Director.

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement