This memo defines a framework for defining content for Netconf. It defines requirements to enable interoperability, extensibility, easy parsing, usability and predictable modelling.
Table of Contents

1. Introduction ............................................. 4
   1.1 Definition of Terms .................................. 4
2. Requirements ............................................. 5
   2.1 Data Modelling Language ............................... 5
   2.2 Conformance ........................................... 5
      2.2.1 Fine Grain Conformance ............................ 5
      2.2.2 Operations on managed objects ..................... 5
      2.2.3 Element Status ................................... 6
      2.2.4 Additional Conformance Information ................ 6
   2.3 Backwards Compatibility ............................... 7
   2.4 Versioning ............................................ 7
   2.5 Keys .................................................. 8
   2.6 Defining Relationships ................................. 8
      2.6.1 Association Relationship ........................... 9
   2.7 Defining Event Messages ............................... 10
   2.8 Considerations for Parse-ability ....................... 11
      2.8.1 Well-formed XML .................................. 11
   2.9 Use an Explicit Namespace on Attributes ............... 12
   2.10 Naming ............................................... 12
   2.11 Error Messages ...................................... 12
   2.12 Schema Documentation ................................ 13
   2.13 Specifying Statistics, Status and Configuration Information ............................. 13
      2.14 Schema Identity Information ........................ 14
3. Modelling Considerations .................................. 15
   3.1 Modularity ............................................ 15
   3.2 Data Types ............................................ 15
   3.3 Elements and Attributes ............................... 15
   3.4 Use Container Elements for Lists ....................... 15
   3.5 Naming implications of using XPATH ..................... 16
      3.5.1 Proper Tag Names .................................. 17
   3.6 Granularity of Data Model ............................. 17
   3.7 Avoid Mixed Content .................................. 17
4. Summary and Example ....................................... 19
   4.1 Summary of Netconf Appinfo Elements & Attributes ...... 19
   4.2 XML Schema for Identity appInfo ........................ 20
   4.3 XML Schema for per element appInfo ..................... 20
   4.4 Managed Object Example ................................ 22
5. Relationship to Netconf Protocol .......................... 26
   5.1 Merge Operation ...................................... 26
   5.2 Replace Operation ..................................... 27
   5.3 Delete Operation ..................................... 28
   5.4 Create Operation ..................................... 29
   5.5 Get Operation ....................................... 30
   5.6 Get Operation with subtree filtering ................... 31
   5.7 Get All Operation .................................... 31
6. Security Considerations ........................................... 33
7. Acknowledgements .................................................. 34
8. References ......................................................... 34
    Authors’ Addresses ............................................... 34
A. Appendix ............................................................. 36
    A.1 Access Control .................................................. 36
        A.1.1 Overview ................................................... 36
        A.1.2 Proposed Solution ......................................... 36
    A.2 Open Issues ..................................................... 37
    Intellectual Property and Copyright Statements .................. 39
1. Introduction

NETCONF [NETCONF-PROTO] can be conceptually partitioned into four layers:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>Configuration data</td>
</tr>
<tr>
<td>Operations</td>
<td>&lt;get-config&gt;, &lt;edit-config&gt;</td>
</tr>
<tr>
<td>RPC</td>
<td>&lt;rpc&gt;, &lt;rpc-reply&gt;</td>
</tr>
<tr>
<td>Application</td>
<td>BEEP, SSH, SSL, console</td>
</tr>
</tbody>
</table>

This document defines a framework for Netconf content. This framework is intended to provide guidance for the creation of Netconf content for the purposes of enabling interoperability, extensibility, parse-ability and usability.

Figure 1

1.1 Definition of Terms

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 [3].

Element: An XML Element[XML].

Managed Entity: A node, which supports netconf[NETCONF] and has access to management instrumentation. This is also referred to as the netconf server.

Managed Object: A collection of one of more Elements that define an abstract thing of interest.
2. Requirements

This section describes some restrictions on Netconf content and the specifications that describe this content, which will increase interoperability between implementations and between different versions of a given implementation.

2.1 Data Modelling Language

XML Schema should be used to define the XML-formatted data that will be transported via Netconf.

2.2 Conformance

When defining netconf content, it is also necessary to define machine-readable conformance for that content. The following are the requirements that have been identified for the conformance and compliance aspects of Netconf data models. This conformance is defined for both the individual elements with the Schema, and also for the entire schema.

Conformance specifies not only whether an object must be supported, but also the level of access, read versus write for example that is minimally required.

2.2.1 Fine Grain Conformance

When defining elements, the "minOccurs" and "maxOccurs" tags MUST be used to specify whether that object is required to have a compliant schema. When defining an attribute, the "use" tag must be used to define whether the attribute is required.

2.2.2 Operations on managed objects

Operations that can be performed on managed objects fall into one of the following equivalence classes: "Create", "Delete", "Read", "Write", and "Execute".

A value of "create" means it is possible to create new instances of this element using commands like the netconf <edit-config> or <copy-config> commands. A value of "delete" means it is possible to destroy instances of this element using commands like the netconf <edit-config>, <copy-config> or <delete-config> operations. A value of "read" means that it is possible to view values of this element using commands like the <get-config>, <get> or <event> operations. A value of "write" means it is possible to modify an instance of this element using commands like the netconf <edit-config> or <copy-config> commands. A value of "execute" means that
there is a side effect execution such as rebooting that is permissible as a result of commands like the netconf <edit-config> or a <copy-config> command or the ability to execute a commands like the <lock>, <unlock> or <kill-session> command.

This memo introduces the appinfo element of "minAccess" and an optional one of "maxAccess" which contain a non-empty list of values. The "minAccess" element defines the set of operations that must be supported in order to claim compliance to this schema. The "maxAccess" element contains the full set of operations that make operational sense for this object. If not present, it assumes the same value as the minAccess tag.

For example, a status object might have a "minAccess" of "read" but a "maxAccess" of "read write" to indicate that it must be possible to perform a get operation the status, but implementations could also allow configuration operations on it as well. In the case of a statistic, both the "minAccess" and "maxAccess" might have values of "read".

<nm:minAcces> <nm:read/> </nm:minAccess>
<nm:maxAcces> <nm:read/> <nm:write/> </nm:maxAccess>

2.2.3 Element Status

As a schema evolves, certain elements may become obsolete. Simply deleting these from the Schema may be acceptable for elements that did not see implementation, but others will require a strategy to allow implementers to migrate away from the old elements.

An optional appinfo element called "status" SHOULD be used to provide the status of the element. When not present, it will assume a value of "current". The other value of this object is "obsolete" which indicates that implementations should consider migrating away from this object and that its implementation is no longer required to be considered conformant. Obsolete content is never removed from the document and its element name can never be re-used

For example
<nm:status> current </nm:status>

2.2.4 Additional Conformance Information

Additional information about conformance should be specified using a
Examples of additional conformance information that may be useful to provide includes how implementations can specify specific exceptions to required conformance, dependencies between elements (in order to do A, you need to first do B) and conditional conformance (if BGP, then ...).

2.2.4.1 Schema Level Conformance

In order to claim compliant Netconf content, all elements MUST conform to their given minOccurs and maxOccurs definitions and all elements with a status of "current" and with a minOccurs greater than or equal to one MUST be supported. In addition, all of the operations listed by the minAccess attribute MUST be supported.

2.3 Backwards Compatibility

Backwards compatibility means that new versions of an XML Schema that defines Netconf Content can be rolled out in a way that does not break existing supporters.

Changes introduced as a result of an update to an existing specification of Netconf content fall into three categories: new concept are added; existing concepts are changed; or existing concepts are obsoleted. Adding new optional content or adding optional new content to the content of a component, such as a new enumeration in a list, are changes that maintain backwards compatibility. Changing the meaning or semantics of existing content, restricting the content of an existing component, or adding or removing required components are changes that do not maintain backwards compatibility.

If an update to an XML Schema is backwards compatibility, then it must use the same element name. A new element name must be used when backwards compatibility is not possible.

2.4 Versioning

Each version of a schema needs to be complete, not a delta from the previous version.

[Editor’s Note: there has been discussion about whether all versions of a Schema maintain backwards compatibility, or if only minor version number changes do. Major version number changes could potentially signify a break in backwards compatibility. For example, version 1.2 would be backwards compatible to version 1.0, while version 2.0 may not be.]
The XML Schema version attribute will be used to signify version

For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  version="3.1">
  <xs:element name="Foo">
  </xs:element>
</xs:schema>
```

This allows applications to be aware of XML Schema versions, and changes to XML Schema version, without forcing instance documents to change to a new schema if the new schema is backwards compatible.

### 2.5 Keys

Keys are an optional construct for specifying the element or set of element that uniquely identifies an instance of a managed object. The XML Schema 'key' construct is used to specify keys.

In the absence of explicitly defined keys, everything can be considered a key from the perspective of the collection of fields that uniquely define an entry.

[Editor’s Note: we may also want to define an attribute called keys so it can be sent over the wire in the instance document.]

```xml
<example to come>
```

Note that being able to query on arbitrary pieces of information provides for multiple views of the managed object, and the optional definition of keys does not preclude this.

### 2.6 Defining Relationships

Relationships between elements come in three forms: associations, extensions and specializations.

An extension to existing element defines information about the same managed object, but just does it in a separate piece of Schema. For example, if a common Schema define information about interfaces, a particular product might define an extension to define information for that interface that is only applicable to that product. To return to our book example, a particular publisher may wish the extend the general book definition to include information specific to
their own books, such as the name of the animal depicted on the cover.

A specialization of an existing element is an extension that only applies to a subset of the instances of the managed object that the original definition applied to. For example, the original element may define information about interfaces, with a specialization being defined that is information only applicable to ATM interfaces. With our book example, a specialization of children’s books might be defined that defines information such as suggested age of reader.

An association exists between two different managed objects. For example, a port is associated with an interface or a book within a bookshelf.

It is important to be able to learn the relationships between the managed objects that are represented in the XML Schema in order to be able to take full advantage of information provided. In addition to this, it is also important to be able to understand these relationships to help predict the behaviour of the system. When a configuration command causes the creation of a managed object represented by a piece of XML schema, it causes the creation of all other bits of XML Schema that represent that managed object as well as any applicable specializations of that object.

Relationships need to be understood in general as well as the specific run-time instances. The first is what is defined in the XML Schema and the second is what we see only over the wire. The general relationship needs to be understood when reading the schema or when designing tools and scripts to use the Schema. For example, interfaces are associated with ports and there is a specific method of learning more about this relationship. The run-time instance relationship, for example that port 3 is associated with interface number 324, or that the Encyclopaedia is on shelf 3 is learned using the general method learned while learning about the generalized relationships.

2.6.1 Association Relationship

The easiest way to define an association relationship is using containment. A book is on a bookshelf, so the following XML makes this relationship obvious and unambiguous

```xml
<bookShelf>
  <self>
    <book/>
  </shelf>
```

Chisholm & Adwankar Expires October 26, 2006 [Page 9]
It is not always possible or desirable to model all associations via containment. Managed objects are often associated with more than one other managed object and containment within both might cause confusion and certainly causes extra XML to be generated. In addition, in some associations, it might not be obvious to decide which objects is contained in which object. And finally, it may be more workable to break the definition of managed objects into smaller, related pieces of XML Schema.

The XML Schema ‘keyref’ construct will be used to define relationships between managed objects.

[Editor’s Note: ensure ‘keyref’ can be used to point at elements not within the same scope. If not, define a similar construct that can be.]

2.7 Defining Event Messages

An event is something that happens which may be of interest. A fault, a change in status, crossing a threshold, or an external input to the system, for example [RFC3877]. Often this results in an asynchronous message, sometimes referred to as a notification or event message, being sent out to interested parties to notify them that this event has occurred. Event messages will be defined in XML Schema with an appinfo of ‘eventClasses’ used to both identify which bits of XML Schema define event messages but also what type of event. An event belongs to one or more classes.

The initial set of classes is fault, information, state, audit, configuration, data, maintenance, metrics, security and heartbeat. A fault event message is generated when a fault condition (error or warning) occurs. An Information event is something that happens of interest which is within the expected operational behaviour and not otherwise covered by another class. A state event indicates a change from one state to another, where a state is a condition or stage in the existence of a managed entity. Audit events provide notification of very specific actions within a managed device. In isolation an audit event provides very limited data. A collection of audit information forms an audit trail. A configuration event, alternatively known as an inventory event, is used to notify that hardware, software, or a service has been added/changed/removed. A
data dump event is an asynchronous event containing information about a system, its configuration, state, etc. A maintenance event signals the beginning, process or end of an action either generated by a manual or automated maintenance action. A metrics event contains a metric or a collection of metrics. This includes performance metrics. A heart beat event is sent periodically to enable testing that the communications channel is still functional.

Note that it may make operational sense to set the minAccess or maxAccess of an element with element classes defined against it to be a null list to indicate that this information can not be read or configured using other commands.

2.8 Considerations for Parse-ability

Netconf content can affect the efficiency and robustness of parsing routines in two ways. The first has to do with whether anything within the Netconf content could be confused with any aspect of the operations, RPC or application protocol layers. If this is possible, then more careful routines need to be written. In particular, it might be difficult to separate out an implementation into separate methods to parse these different layers if it is necessary to parse the Netconf content and match open and close brackets rather than just looking for an appropriate close tag.

Another aspect where the content will affect performance of the parsing routines is on the assumptions that the parsing routine can make. The following section outlines some restrictions on the Netconf content that will positively impact the performance of parsing routines with minimum impact on the usability of the solution.

2.8.1 Well-formed XML

All XML used within a Netconf solution needs to be well formed [XML] and conform to an XML Schema specification that conforms to the guidelines in this document.
2.9 Use an Explicit Namespace on Attributes

All attributes should be prefixed so that they belong to a specific namespace. This encourages meaningful definitions that are free of collisions.

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

2.10 Naming

All NetMod base elements SHOULD be defined in the namespace urn:ietf:params:xml:ns:netmod:base:1.0

All NetMod defined datatypes SHOULD be defined in the namespace urn:ietf:params:xml:ns:netmod:datatypes:1.0

The name of an element SHOULD be unique in a given namespace and should be addressed in a reference to a given namespace.

2.11 Error Messages

Within Netconf, the `<error-app-tag>` element is used to provide application-level error codes. If implementations don’t understand the application-level specific error codes, they still have the generic one to go by, but the application-level specific error codes can provide information about the specific problem that has happened.

A non-exhaustive set of error messages that may get generated by the application as a result of performing netconf operations against that data model is included within the XML Schema that defines the Netconf content.

[Editor’s Note: The definition of the appErrors appinfo could in theory be made richer than it is, so long as the information still goes over the wire as specified in the base netconf specification. We could specify which of the operation equivalence classes can trigger this message (read, for example) as well as any additional fields that should get reported in the message. Note that we can’t mix content on the wire, so any additional fields would need to be embedded - "I can’t read Les Miserable" as appose to "I can’t read <title>Les Miserable</title>". ]
An optional appinfo called ‘appErrors’ is used to specify these application-level error messages.

```xml
<nm:appErrors>
  <nm:appError>
    Book is in language you do not understand.
  </nm:appError>
</nm:appErrors>
```

These are applicable to any element.

[Editor’s Note: How closely tied are these to the known set of operations that can be performed on the data? How is this determined?]

### 2.12 Schema Documentation

The "documentation" tag must be used to provide all additional information to implementers and users of a schema that cannot be modelled within the schema itself or using the appinfo defined within this memo. This includes further restrictions and additional complexities as well as any information that will be helpful in understanding the element.

Note that other means of documenting, including the <!-- --> construct are not as easily associated within specific elements and not necessarily understood by all tools.

### 2.13 Specifying Statistics, Status and Configuration Information

[Editors Note: What about historical performance statistics?]

There may be potential value in being able to easily distinguish between configuration, status and statistical information within a data model. This would allow better understanding of nature of each piece of information without requiring specific knowledge of the context.

Propose adding an optional appinfo called dataType which takes a value of ‘configuration’, ‘statistics’, or ‘status’.

[Editor’s Note: rework these examples to use appinfo]

For example
<nm:dataType>configuration</nm:dataType>

[Editor’s Note: Test on existing data models?]

2.14 Schema Identity Information

Replacing the following with something from Dublin Core (http://dublincore.org) is for further study.

A schema must contain a schema Identity annotation as part of the appinfo on the highest level schema element. The Identity provides schema information such as last update of this schema, organization, contact, description and revision history. The contact and revision information can occur multiple times in a schema identity.

An example of identity element is as given below

<xs:appinfo>
    <nm:Name>NetConf State Schema</nm:Name>
    <nm:LastUpdated>2001-12-17T09:30:47-05:00</nm:LastUpdated>
    <nm:Organization>Example.com</nm:Organization>
    <nm:Contact>
      <nm:Name>ExampleName</nm:Name>
      <nm:Postal>ExampleAddress</nm:Postal>
      <nm:EMail>http://www.example.com</nm:EMail>
      <nm:Phone>0000000</nm:Phone>
    </nm:Contact>
    <nm:Description>The Description of NetConf State Schema</nm:Description>
    <nm:RevisionHistory>
      <nm:Revision>2004-12-17T09:30:47-05:00</nm:Revision>
      <nm:Description>The first version of the schema</nm:Description>
    </nm:RevisionHistory>
  </nm:Identity>
</xs:appinfo>
3. Modelling Considerations

3.1 Modularity

It is better to publish Netconf content as a series of XML Schema rather than as a single monolithic data model.

3.2 Data Types

XML Schema has 44 built in data types [XML-SCHEMA-2]. Potentially reusable data types should be declared as simple or complex type, rather than element.

Emphasis should be replaced on creating reusable application-level data types such as IP addresses, DateAndTime or OSI states, rather than developing 20 different flavours of integers.

3.3 Elements and Attributes

When designing encoding rules for NETCONF content, the following guidelines should be used to determine when use of elements is appropriate and when is use of attributes.

Attributes should contain metadata about the element, not true data. Therefore, information about the managed entity should not be encoded in attributes.

In addition, attributes are unordered, can appear only once, and can have no children. When modelling data in an XML Schema, it is important to leave room for future expansion - in future specifications or future software releases. This is another reason to only use attributes for metadata.

3.4 Use Container Elements for Lists

A distinct container should be used when encoding lists with multiple instances (maxOccurs > 1), use a distinct container element. Sometimes this will be the plural form of the instance element, but often there is a more natural container available. Some containers exist in the ‘real world’ and some are only modelling artefacts. When an appropriate real-world container is available, it is not necessary to create a modelling artefact to artificially contain the list.
In this example, the element ‘book’ is contained within the "books" element.

```xml
<books>
    <book>....</book>
    <book>....</book>
    <book>....</book>
</books>

<bookshelf>
    <book>....</book>
    <book>....</book>
    <book>....</book>
    <book>....</book>
</bookshelf>
```

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

### 3.5 Naming implications of using XPATH

XPath [XPATH] can be used to locate managed objects in a given namespace. XPATH based addressing can also be used to select a set of managed objects based on a set of criteria’s, select content that is combination of different managed object values and to create simple expressions of managed objects.

Examples of XPATH based addressing are shown below:


2. Provide ACL information of the . /bk:/ACL


5. Select all books that have "NetMod" in their description and average review rating is greater than 4. //bk:book[(contains(bk:bookTitle/@bk:value,’NetMod’)) and (bk:AverageReview/@bk:value>'4')]}

6. Find number of books whose publication year is greater than 2003. count(//bk:book[bk:PublicationYear/@bk:value>'2003'])
3.5.1 Proper Tag Names

When choosing element names, they should:

- use ASCII (7-bit)
- be lower camel, a method of writing compound words or phrases where the words are joined without spaces, and each word is capitalized within the compound. The first letter of the compound is lower case. An example would be lowerCamel.
- Whenever possible, use full words. There are some well-known abbreviations and short forms, such as "config" that would be considered acceptable
  - Should be consistent with existing vocabularies

These are guidelines only and should be considered secondary to the need for consistency with existing vocabularies. For example, when encoding SNMP MIB variables names in NETCONF, use the existing names (ifAddr) instead of shifting to these guidelines (ifAddress). 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.

3.6 Granularity of Data Model

Designers should give some thought to the high level information they users need to manage the device and not simple expose the low level information that they have available. Ideally, it should be possible to make a small change to the data model and have it trigger a big change in the managed entity.

3.7 Avoid Mixed Content

Mixed content is defined as elements that can contain both data and other elements. Elements in NETCONF 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>
4. Summary and Example

4.1 Summary of Netconf Appinfo Elements & Attributes

The following table summarizes the XML Schema appinfo introduced in this memo. When these appinfo are used in the definition of XML Schema for use with netconf, they are applicable to all instances of that Schema.

<table>
<thead>
<tr>
<th>appinfo item</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>minAccess</td>
<td>Mandatory</td>
</tr>
<tr>
<td>maxAccess</td>
<td>Optional</td>
</tr>
<tr>
<td>status</td>
<td>Optional</td>
</tr>
<tr>
<td>appErrors</td>
<td>Recommended</td>
</tr>
<tr>
<td>identity</td>
<td>Mandatory</td>
</tr>
<tr>
<td>eventClass</td>
<td>Optional</td>
</tr>
<tr>
<td>dataType</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Figure 17
4.2 XML Schema for Identity appInfo

```xml
<xs:complexType name="IdentityType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="LastUpdated" type="xs:dateTime"/>
    <xs:element name="Organization" type="xs:string"/>
    <xs:element name="Contact" type="ContactType" maxOccurs="unbounded"/>
    <xs:element name="Description" type="Description" minOccurs="0"/>
    <xs:element name="RevisionHistory" type="RevisionType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="RevisionType">
  <xs:sequence>
    <xs:element name="Revision" type="xs:dateTime"/>
    <xs:element name="Description" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ContactType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Postal" type="xs:string" minOccurs="0"/>
    <xs:element name="EMail" type="xs:anyURI"/>
    <xs:element name="Phone" type="xs:string" minOccurs="0"/>
    <xs:element name="otherInfo" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

4.3 XML Schema for per element appInfo

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns:netmod:base:1.0"
  targetNamespace="urn:ietf:params:xml:ns:netmod:base:1.0"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="0.1">
  <xs:complexType name="AppInfo">
    <!-- Schema content -->
  </xs:complexType>
</xs:schema>
```
<xs:annotation>
  <xs:documentation>
  This is a set of information that can be applied to any
  element.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="minAccess" type="AccessList"/>
  <xs:element name="maxAccess" type="AccessList"/>
  <xs:element name="status" type="Status" minOccurs="0"/>
  <xs:element name="appErrors" type="AppErrors" minOccurs="0"/>
  <xs:element name="eventClasses" type="EventClasses" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

<xs:element name="appInfo" type="AppInfo"/>

<xs:simpleType name="Status">
  <xs:restriction base="xs:string">
    <xs:enumeration value="current"/>
    <xs:enumeration value="obsolete"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="AccessList">
  <xs:sequence>
    <xs:element name="access" type="AccessType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AccessType"/>
<xs:element name="Access" type="AccessType" abstract="true"/>
<xs:element name="read" type="AccessType" substitutionGroup="Access"/>
<xs:element name="write" type="AccessType" substitutionGroup="Access"/>
<xs:element name="create" type="AccessType" substitutionGroup="Access"/>
<xs:element name="delete" type="AccessType" substitutionGroup="Access"/>
<xs:element name="execute" type="AccessType" substitutionGroup="Access"/>
<xs:complexType name="AppErrors">
<xs:sequence>
    <xs:element name="appError" type="xs:string" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="EventClassType"/>
<xs:element name="EventClass" type="EventClassType" abstract="true"/>
<xs:element name="fault" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="information" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="state" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="configuration" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="data" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="maintenance" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="metrics" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="security" type="EventClassType" substitutionGroup="EventClass"/>
<xs:element name="heartbeat" type="EventClassType" substitutionGroup="EventClass"/>

<xs:complexType name="EventClasses">
    <xs:sequence>
        <xs:element name="class" type="EventClassType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

4.4 Managed Object Example

An example of a node that describes a system description is shown below.

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
This element defines information about books.

</xs:element>
<xs:element name="Author" type="xs:string">
  <xs:annotation>
    <xs:documentation>
    The author of this book.
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:minAccess> <nm:read/> </nm:minAccess>
        <nm:maxAccess> <nm:read/> <nm:create/>
        <nm:delete/> <nm:write/> </nm:maxAccess>
        <nm:status> current </nm:status>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:element name="PublicationYear" type="xs:string">
  <xs:annotation>
    <xs:documentation>
    The year this book was published
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:minAccess> <nm:read/> </nm:minAccess>
        <nm:maxAccess> <nm:read/> <nm:create/>
        <nm:delete/> <nm:write/> </nm:maxAccess>
        <nm:status> current </nm:status>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:element name="LastIssueDate" type="xs:string">
  <xs:annotation>
    <xs:documentation>
    The last issue date for this book.
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:minAccess> <nm:read/> </nm:minAccess>
        <nm:maxAccess> <nm:read/> <nm:create/>
        <nm:delete/> <nm:write/> </nm:maxAccess>
        <nm:status> current </nm:status>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:element name="AverageReview" type="xs:string">
  <xs:annotation>
  </xs:annotation>
</xs:element>
An example of an instance of a book node is

```
  <bookTitle>XYZ</bookTitle>
  <Author>ABC</Author>
  <PublicationYear>2005</PublicationYear>
  <LastIssueDate>2005-02-02</LastIssueDate>
  <AverageReview>5</AverageReview>
</book>
```
5. Relationship to Netconf Protocol

The Netconf architecture supports a clear separation between content and protocol. This is an important architectural separation that should be maintained. That having been said, there are major advantages to ensuring that the content of Netconf is well behaved and predictable.

Whether a Netconf implementation can be said to be compliant without also being compliant to the guidelines within this memo is an area of further study.

The following examples illustrate the netconf protocol commands being executed against netmod compliant Schemas.

5.1 Merge Operation

RPC Request:
<nc:rpc message-id="101"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<nc:edit-config>
<nc:target>
<nc:running/>
</nc:target>
<nc:config nc:operation="replace">
<nc:config nc:operation="merge">
<bookTitle>NetMod for Dummies: Part2</bookTitle>
<Author>NetMod Experts</Author>
/PublicationYear>2003</PublicationYear>
/LastIssueDate>2003-10-15</LastIssueDate>
/AverageReview>1</AverageReview>
</book>
</nc:config>
</nc:edit-config>
</nc:rpc>

RPC Reply:
<nc:rpc-reply message-id="101"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<nc:ok/>
</nc:rpc-reply>
5.2 Replace Operation

RPC Request:
```xml
<nc:rpc message-id="101"
   xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:default-operation>replace</nc:default-operation>
    <nc:config nc:operation="replace">
        <bookTitle>NetMod for Dummies: Part2</bookTitle>
        <Author>NetMod Experts</Author>
        <PublicationYear>2003</PublicationYear>
        <LastIssueDate>2003-10-15</LastIssueDate>
        <AverageReview>1</AverageReview>
      </book>
    </nc:config>
  </nc:edit-config>
</nc:rpc>
```

RPC Reply:
```xml
<nc:rpc-reply message-id="101"
   xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:ok/>
</nc:rpc-reply>
```
5.3 Delete Operation

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

RPC Reply:
<nc:rpc-reply message-id="101"
 xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:ok/>
</nc:rpc-reply>
5.4 Create Operation

RPC Request :

<nc:rpc message-id="101"
   xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:default-operation>replace</nc:default-operation>
    <nc:config nc:operation="create">
        <bookTitle>NetMod for Dummies: Part6</bookTitle>
        <Author>NetMod Experts</Author>
        <PublicationYear>2003</PublicationYear>
        <LastIssueDate>2003-10-15</LastIssueDate>
        <AverageReview>1</AverageReview>
      </book>
    </nc:config>
  </nc:edit-config>
</nc:rpc>

RPC Reply :

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

RPC Request:
<nc:rpc message-id="101"
   xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:get-config>
    <nc:source>
      <nc:running/>
    </nc:source>
  </nc:get-config>
</nc:rpc>

RPC Reply:
<nc:rpc-reply message-id="101"
   xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:data>
      <bookTitle>NetMod for Dummies: Part2</bookTitle>
      <Author>NetMod Experts</Author>
      <PublicationYear>2003</PublicationYear>
      <LastIssueDate>2003-10-15</LastIssueDate>
      <AverageReview>1</AverageReview>
    </book>
  </nc:data>
</nc:rpc-reply>
5.6 Get Operation with subtree filtering

RPC Request :
<nc:rpc message-id="101"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:get-config>
    <nc:source>
      <nc:running/>
    </nc:source>
  </nc:get-config>
</nc:rpc>

RPC Reply :
<nc:rpc-reply message-id="101"
              xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:data>
      0596002924
    </ISBN>
  </nc:data>
</nc:rpc-reply>

5.7 Get All Operation

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

RPC Reply :
<nc:rpc-reply message-id="101"
               xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:data>
    <configs xmlns="urn:ietf:params:xml:ns:netconf:state:1.0">
      <config>
        <lock-status>
          <lock-state>locked</lock-state>
          <locked-by>7</locked-by>
        </lock-status>
      </config>
    </configs>
  </nc:data>
</nc:rpc-reply>
<config-name>
  <startup/>
</config-name>
<config>
  <lock-status>
    <lock-state>unlocked</lock-state>
    <locked-by/>
  </lock-status>
  <config-name>
    <candidate/>
  </config-name>
</config>
<config>
  <lock-status>
    <lock-state>unlocked</lock-state>
    <locked-by/>
  </lock-status>
  <config-name>
    <startup/>
  </config-name>
</config>
</configs>
<state xmlns="urn:ietf:params:xml:ns:netconf:state:1.0">
  <sessions>
    <session>
      <session-id>6</session-id>
      <userName>Bob</userName>
      <login-time>2:56:01 PM</login-time>
    </session>
  </sessions>
</state>
<nc:config>
    <bookTitle>NetMod for Dummies: Part2</bookTitle>
    <Author>NetMod Experts</Author>
    <PublicationYear>2003</PublicationYear>
    <LastIssueDate>2003-10-15</LastIssueDate>
    <AverageReview>1</AverageReview>
  </book>
</nc:config>
</nc:data>
</nc:rpc-reply>
6. Security Considerations

To be determined once specific aspects of this solution are better understood. In particular, the access control framework and the choice of transport will have a major impact on the security of the solution.
7. Acknowledgements

This document is a result of discussions at IETF 59, 60 and 64, as well as on the mailing list by the following people: Sharon Chisholm, David Harrington, Ray Atarashi, Yoshifumi Atarashi, Bert Wijnen, Dan Romascu, Andy Bierman, Randy Presuhn, Chris Lonvick, Eliot Lear, Avri Doria, Juergen Schoenwaelder, Rob Enns, Faye Ly, Andre Westerinen, Orly Nicklass, Alexander Clemm, Keith McCloghrie, Hector Trevino, James Balestriere Simon Leinen and Ladislav Lhotka.

8. References


[refs.RFC2119]  Bradner, s., "Key words for RFCs to Indicate Requirements Levels", RFC 2119, March 1997.


Authors’ Addresses

Sharon Chisholm
Nortel
3500 Carling Ave
Nepean, Ontario  K2H 8E9
Canada

Email: schishol@nortelcom
Sandeep Admankar
Motorola
1301 East Algonquin Road
MS IL02-2240
Schaumburg, Il  60196
USA

Email: sandeep.adwankar@motorola.com
Appendix A. Appendix

A.1 Access Control

A.1.1 Overview

For discussing access control, we shall use the same equivalence classes for operations on managed objects: "Create", "Delete", "Read", "Write", and "Execute".

Access control is defined only for the node it is associated with. It does not cascade throughout a containment hierarchy.

There are four classes of access control that can possibly be defined:

1. Resource Category: High Level like BGP or DNS, similar to syslog concept
2. Resource Type: Lowest level before instance
3. Resource Instance
4. Operation Type

There is a fifth class that enables access control within an instance to specific values. This level of control is beyond the scope of this specification.

[Editor’s Note: What about permission to traverse through a hierarchy? This naturally imparts information about the hierarchy that might allow a user to gleam things they should not.

The two main use cases for consideration are single customer and multiple customers on a single box. Note that the concept of a virtual router has been successful in solving the latter problem in SNMP and CLI.

Ideally the security model used in netconf is integratable with well-deployed solutions like RADIUS and TACACS+.

Netconf may only need a mechanism to report information about access control and an API to enable easy integration to an existing solution.

A.1.2 Proposed Solution

[Editors Note: This area is for further study. Which bits belong in
the data model and which in the protocol? Can define a simple
security model that modelled on what CLIs do and integrate with
radius? What impacts would this have on the data model? Do we want
to a fully exposed and described model or just the ability to answer
the question "Is user A allows to run this command?". How do we
identify users, or do we need to (is it out of scope)? The
requirement is to identify it so it can be deployed in deployed
security infrastructure.]

A solution will need to identify the type off access that is
permitted, profile an identifier for this instance of access
permissions and potentially identify a timeframe that this access is
good for. This access would then need to be associated with
particular class and instance data, as described above.

    <xs:complexType name="ACLEntryType">
    <xs:sequence>
      <xs:element name="access" type="AccessType"/>
      <xs:element name="Identifier" type="IdentifierType"/>
      <xs:element name="accessDuration"
        type="xs:duration" minOccurs="0"/>
    </xs:sequence>
    </xs:complexType>

A.2 Open Issues

What happens when we are importing from other organizations, and they
have different rules. This needs to be thought through for version,
backwards compatibility, mixed model, etc to ensure this won’t cause
any problems.

Do we actually needed minAccess and maxAccess or can it be deleted.
It was suggested that it could be added back later if it was
determined to be necessary. Some people felt that conformance, even
run-time indication of what is actually supported, was important.
Others felt it didn’t work well in SNMP, so perhaps native minOccurs/
maxOccurs would be sufficient. It was suggested that it might be
more interesting to distinguish between state, statistics and
configuration when defining data rather than providing min and max
access.

Around section 2.5.1, there was discussion about whether keys could
change over time (across reboots, if cards moved, etc) and if so, if
perhaps some other less meaningful but more permanent index was also
useful. There were some concerns raised about the usability of these
sort of arbitrary number indices. There was also discussion about the difference between internal and external identifiers – those generated by the system and those input from external systems. The latter will not change unless the user actually wants it to change.
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights 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; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat 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 implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.
Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.