Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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 May 21, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).
Abstract

This memo defines a framework for defining content for NETCONF using XML Schema. 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 ................. 6
    2.2.3. Element Status .................................. 6
    2.2.4. Additional Conformance Information ............ 7
  2.3. Backwards Compatibility ........................... 7
  2.4. Versioning .......................................... 8
  2.5. Keys ................................................ 8
  2.6. Defining Relationships ............................ 9
    2.6.1. Association Relationship ....................... 10
  2.7. Defining Notification Event Messages .............. 11
  2.8. Considerations for Parse-ability .................. 12
    2.8.1. Well-formed XML ................................ 13
  2.9. Naming .............................................. 13
  2.10. Error Messages .................................... 13
  2.11. Schema Documentation ............................ 14
  2.12. Specifying Statistics, Status and Configuration
       Information ........................................ 14
3. Modelling Considerations ............................... 15
  3.1. Modularity ........................................ 15
  3.2. Data Types ......................................... 15
  3.3. Elements and Attributes ........................... 15
  3.4. Naming implications of using XPATH ............... 15
    3.4.1. Proper Tag Names ................................ 16
  3.5. Granularity of Data Model ........................ 16
  3.6. Avoid Mixed Content ............................... 17
4. Summary and Example .................................... 18
  4.1. Summary of NETCONF Appinfo Elements & Attributes .. 18
  4.2. XML Schema for per element appInfo ............... 18
  4.3. Managed Object Example ........................... 18
5. Relationship to NETCONF Protocol ....................... 24
6. Security Considerations ................................ 25
7. Acknowledgements ....................................... 26
8. Normative References ................................... 27

Appendix A. Interworking with Yang .......................... 28
1. Introduction

[NETCONF] 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; &lt;notification&gt;</td>
</tr>
<tr>
<td>RPC</td>
<td>&lt;rpc&gt;, &lt;rpc-reply&gt;</td>
</tr>
<tr>
<td>Transport</td>
<td>BEEP, SSH, SSL, console</td>
</tr>
</tbody>
</table>

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

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 [RFC2119].

Element: An XML Element [XML].

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

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

This section describes some restrictions on NETCONF content and provides guidance on how to use XML Schema to define 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.

XML Schema is a great choice for specifying NETCONF content because it is an existing, well-defined language which allows the NETCONF community to get immediately started defining NETCONF content. The language can be used as is and NETCONF specific extensions can be added as appinfo extensions to the specifications when they are sufficiently mature.

Other benefits of using XML Schema result from the fact it is widely used within the data communications industry and other industries. This means that the NETCONF community can take advantage of existing tools, training and subject matter experts for dealing with most aspects of the content.

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 to 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 the command. For example, 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:minAccess> <nm:read/> </nm:minAccess>
<nm:maxAccess> <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 documentation tag.

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.

The XML Schema version attribute must 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. Elements whose content can be created, modified or deleted MUST specify keys.
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xsl:element name="Books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="book" type="BookType"
        minOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xsl:element>

<xsl:complexType name="BookType">
  <xs:sequence>
    <xs:element name="title" type="xs:string" />
    <xs:element name="best-quote" type="xs:string" />
  </xs:sequence>
</xs:complexType>

</xs:schema>

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 defines 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 stores 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>
  <shelf>
    <book/>
  </shelf>
</bookShelf>
```
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.

2.7. Defining Notification Event Messages

NETCONF provides a mechanism to send asynchronously event messages [NETCONF-EVENT]. The protocol specification makes few restrictions on the content or source of NETCONF notifications. The only requirement is that the all notifications contain the <eventTime> element. The additional content which will be sent asynchronously via NETCONF notifications should be clearly identified within the XML Schema so that NETCONF clients will know what behaviour to expect.

Event messages will be defined in XML Schema with an appinfo of ‘eventClasses’ used to identify the 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.
The eventClass is specified in the appInfo of the notification definition and also sent over the wire. In order to be sent over the wire, it needs to be explicitly defined as part of the notification content.

The following example definition of a notification demonstrates how to have a single definition of content for both notifications and other NETCONF operations.

```xml
<xs:element name="bookCheckout"
    type="book:BookCheckoutType"
    substitutionGroup="ncEvent:notificationContent">
    <xs:annotation>
        <xs:documentation>
            When a book is checked out, a notification containing
            this information is sent to interested parties. It can
            also be read via a get operation.
        </xs:documentation>
        <xs:appinfo>
            <nm:appinfo>
                <nm:eventClass><nm:informational/></nm:eventClass>
                <nm:minAccess><read/></nm:minAccess>
            </nm:appinfo>
        </xs:appinfo>
    </xs:annotation>
</xs:element>

<xs:complexType name="BookCheckoutType">
    <xs:complexContent>
        <xs:extension base="ncEvent:NotificationContentType">
            <xs:sequence>
                <xs:element name="eventClass"/>
                <xs:element name="status"/>
                <xs:element name="whatHappened"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

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 as NETCONF content needs to be well formed [XML] and conform to an XML Schema specification that conforms to the guidelines in this document.

2.9. Naming

All NETCONF content base elements SHOULD be defined in the namespace urn:ietf:params:xml:ns:netmod:[something]

2.10. 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 should be 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?]

2.11. 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.

2.12. Specifying Statistics, Status and Configuration Information

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.

This memo adds an optional appinfo called dataType which takes a value of ‘configuration’, ‘statistics’, or ‘status’.

For example

```xml
<nm:dataType>configuration</nm:dataType>
```
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 XML Schema File [XSD].

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.

In addition, using types instead of declaring everything to be an element aids extensibility. An example of where this was done is in the NETCONF Notification [NETCONF-EVENT] draft in the Notification Management Schema with the <Netconf> root element.

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 using of attributes is.

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

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 why it is important to only use attributes for metadata.

3.4. 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, 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:


4. 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')]}

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

3.4.1. 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.

3.5. 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.

3.6. 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. They must not contain both.

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.

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

<not-valid>
  <element data><more data>maybe some</more></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>eventClass</td>
<td>Optional</td>
</tr>
<tr>
<td>dataType</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Figure 2

4.2. XML Schema for per element appInfo

4.3. Managed Object Example

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:nm="urn:ietf:params:xml:ns:netmod:base:1.0"
xmlns:ncEvent="urn:ietf:params:xml:ns:netconf:notification:1.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.1">
  <xs:import namespace="urn:ietf:params:xml:ns:netconf:notification:1.0"
               schemaLocation="http://www.iana.org/assignments/xml-registry/schema/notification.xsd"/>
</xs:schema>
```
<xs:element name="Books">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="book" type="book:BookType" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:key name="BookKey">
        <xs:selector xpath="Books/book"/>
        <xs:field xpath="@title"/>
    </xs:key>
</xs:element>

<xs:complexType name="BookType">
    <xs:annotation>
        <xs:documentation xml:lang="en">This element defines information about books</xs:documentation>
        <xs:appinfo>
            <nm:appinfo>
                <nm:minAccess> <nm:read/> </nm:minAccess>
                <nm:maxAccess> <nm:read/> <nm:write/> </nm:maxAccess>
                <nm:status> current </nm:status>
                <nm:appErrors>
                    <nm:appError>Book lent out</nm:appError>
                    <nm:appError>Book not interesting</nm:appError>
                    <nm:appError>Book in language you don’t know</nm:appError>
                    <nm:appError>Book eaten by book worm</nm:appError>
                </nm:appErrors>
            </nm:appinfo>
        </xs:appinfo>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="title" type="book:BookTitle" />
        <xs:element name="Author" type="book:BookAuthor" />
        <xs:element name="PublicationYear" type="book:BookPublishedDate" />
        <xs:element name="AverageReview" type="book:BookReviewScore" minOccurs="0" />
        <xs:element name="best-quote" type="xs:string"
<xs:complexType name="Book">
    <xs:complexContent>
        <xs:restriction base="xs:sequence">
            <xs:element minOccurs="0" name="title">
                <xs:simpleType name="BookTitle">
                    <xs:annotation>
                        <xs:documentation>
                        The title of the book
                        </xs:documentation>
                        <xs:appinfo>
                            <nm:appinfo>
                                <nm:minAccess> <nm:read/></nm:minAccess>
                                <nm:maxAccess> <nm:read/> </nm:maxAccess>
                            </nm:appinfo>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:simpleType>
            </xs:element>
            <xs:element minOccurs="0" name="isbn">
                <xs:simpleType name="BookISBN">
                    <xs:annotation>
                        <xs:documentation>
                        The ISBN of the book
                        </xs:documentation>
                        <xs:appinfo>
                            <nm:appinfo>
                                <nm:minAccess> <nm:read/></nm:minAccess>
                                <nm:maxAccess> <nm:read/> </nm:maxAccess>
                            </nm:appinfo>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:simpleType>
            </xs:element>
            <xs:element minOccurs="0" name="author">
                <xs:simpleType name="BookAuthor">
                    <xs:annotation>
                        <xs:documentation>
                        The author of the book
                        </xs:documentation>
                        <xs:appinfo>
                            <nm:appinfo>
                                <nm:minAccess> <nm:read/></nm:minAccess>
                                <nm:maxAccess> <nm:read/> </nm:maxAccess>
                            </nm:appinfo>
                        </xs:appinfo>
                    </xs:annotation>
                </xs:simpleType>
            </xs:element>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
<xs:union memberTypes="xs:string"/>
</xs:simpleType>

<xs:simpleType name="BookPublishedDate">
  <xs:annotation>
    <xs:documentation>
The date of publication for the book
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:minAccess> <nm:read/> </nm:minAccess>
        <nm:maxAccess> <nm:read/> </nm:maxAccess>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:union memberTypes="xs:dateTime"/>
</xs:simpleType>

<xs:simpleType name="BookReviewScore">
  <xs:annotation>
    <xs:documentation>
The average review score for the book
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:minAccess> <nm:read/> <nm:write/> </nm:minAccess>
        <nm:maxAccess> <nm:read/> <nm:write/> </nm:maxAccess>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="one-star"/>
    <xs:enumeration value="two-star"/>
    <xs:enumeration value="three-star"/>
    <xs:enumeration value="four-star"/>
    <xs:enumeration value="five-star"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="bookCheckouts">
  <xs:complexType>
    <xs:sequence>
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="bookCheckout"
    type="book:BookCheckoutType"
    substitutionGroup="ncEvent:notificationContent">
  <xs:annotation>
    <xs:documentation>
      When things just happen, this notification is sent.
      Note that this information is sent as a notification
      when it is sent, but can also be retrieved via a get
      operation.
    </xs:documentation>
    <xs:appinfo>
      <nm:appinfo>
        <nm:eventClass><nm:informational/></nm:eventClass>
        <nm:minAccess><read/></nm:minAccess>
      </nm:appinfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:complexType name="BookCheckoutType">
  <xs:complexContent>
    <xs:extension base="ncEvent:NotificationContentType">
      <xs:sequence>
        <xs:element name="eventClass"/>
        <xs:element name="status"/>
        <xs:element name="whatHappened"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
An example of an instance of a book node is

```xml
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <book>
    <title>NETCONF for Geniuses</title>
    <Author>Netty Man</Author>
    <PublicationYear>2007-07-08T00:01:00Z</PublicationYear>
    <AverageReview>five-star</AverageReview>
  </book>
</Books>
```
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.
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 Romascanu, Andy Bierman, Randy Presuhn, Chris Lonvick, Eliot Lear, Avri Doria, Juergen Schoenwaelder, Rob Ennes, Faye Ly, Andre Westerinen, Orly Nicklass, Alexander Clemm, Keith McCloghrie, Hector Trevino, James Balestriere Simon Leinen and Ladislav Lhotka. The editor would also like to thank Sandeep Admankar for his work on earlier versions of this document.
8. Normative References


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


Appendix A. Interworking with Yang

Yang is a proposal for specifying NETCONF content which defines its own syntax. The authors of Yang have discussed automating a translation from Yang to XSD. Their original view would be of a translation that only uses core XSD syntax, and would therefore result in a loss of features. This section describes a way of getting to a non-lossy mapping from Yang into XSD.

Many of the features described within the XML Schema [XSD] specification and the body of this memo can be used in a mapping from Yang. To get a more complete mapping, the following changes should be made:

- Add an additional keyref definition as nm:keyref without the scope limitations of the native keyref definition.

- Add a status of ‘depricated’ to the list of status values. Note that this value was not added to the body of this memo since it was felt that in practice people only used two status levels for objects.

- Map the yang ‘config’ boolean into the datatype appInfo. Note though, that the datatype appInfo has two types of non-configuration data, status and statistics so this gap in Yang would need to be addressed.

The above list is from a cursory review of the Yang specification. Additional changes may be deemed necessary after a more thorough review.
Author’s Address

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

Email: schishol@nortel.com