NETMOD	L. Lhotka
Internet-Draft	CESNET
Intended status: Standards Track	R. Mahy
Expires: September 9, 2009	Plantronics
	S. Chisholm
	Nortel
	March 08, 2009

Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content
draft-ietf-netmod-dsdl-map-01

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and 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 September 9, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This draft describes the mapping rules for translating YANG data models into XML schemas using Document Schema Definition Languages (DSDL) and outlines the procedure for validating various types of NETCONF protocol data units using these schemas.

Table of Contents

1. Introduction

TOC

The NETCONF Working Group has completed a base protocol used for configuration management [1] (Enns, R., “NETCONF Configuration Protocol,” December 2006.). This base specification defines protocol bindings and an XML container syntax for configuration and management operations, but does not include a modeling language or accompanying rules for how to model configuration and status information (in XML syntax) carried by NETCONF. The IETF Operations Area has a long tradition of defining data for SNMP Management Information Bases (MIBs) [2] (Case, J., Fedor, M., Schoffstall, M., and J. Davin, “Simple Network Management Protocol (SNMP),” May 1990.) using the SMI language [3] (McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., “Structure of Management Information Version 2 (SMIv2),” April 1999.) to model its data. While this specific modeling approach has a number of well-understood problems, most of the data modeling features provided by SMI are still considered extremely important. Simply modeling the valid syntax rather than additional semantic relationships has caused significant interoperability problems in the past.

The NETCONF community concluded that a data modeling framework is needed to support ongoing development of IETF and vendor-defined management information modules. The NETMOD Working Group was chartered to address this problem, by defining a new human-friendly modeling language based on SMIng [4] (Elliott, C., Harrington, D., Jason, J., Schoenwaelder, J., Strauss, F., and W. Weiss, “SMIng Objectives,” December 2001.) and called YANG [5] (Bjorklund, M., Ed., “YANG - A data modeling language for NETCONF,” March 2009.).

Since NETCONF uses XML for encoding its protocol data units (PDU), it is natural to express the constraints on NETCONF content using standard XML schema languages. For this purpose, the NETMOD WG selected the Document Schema Definition Languages (DSDL) that is being standardized as ISO/IEC 19757 [6] (ISO/IEC, “Document Schema Definition Languages (DSDL) - Part 1: Overview,” 11 2004.). The DSDL framework comprises a set of XML schema languages that address grammar rules, semantic constraints and other data modeling aspects but also, and more importantly, do it in a coordinated and consistent way. While it is true that some DSDL parts have not been standardized yet and are still work in progress, the three parts that the YANG-to-DSDL mapping relies upon - RELAX NG, Schematron and DSRL - already have the status of an ISO/IEC International Standard and are supported in a number of software tools.

This document contains the specification of a mapping that translates YANG data models to XML schemas utilizing a subset of the DSDL schema languages. The mapping procedure is divided into two steps: In the first step, the structure of the data tree, RPC signatures and notifications is expressed as a single RELAX NG grammar with simple annotations representing additional data model information (metadata,

documentation, semantic constraints, default values etc.). The second step then generates a coordinated set of DSDL schemas that can validate specific XML documents such as client requests, server responses or notifications, perhaps also taking into account additional context such as active capabilities.
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 [7] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).
In the text, we also use the following typographic conventions:

- *YANG statement keywords are delimited by single quotes.
- *Literal values are delimited by double quotes.
- *XML element names are delimited by "<" and ">" characters.
- *Names of XML attributes are prefixed by the "@" character.

XML elements names are always written with explicit namespace prefixes corresponding to the following XML vocabularies:

"a" DTD compatibility annotations [8] (Clark, J., Ed. and M. Murata, Ed., “RELAX NG DTD Compatibility,” December 2001.)

"dc" Dublin Core metadata elements [9] (Kunze, J., “The Dublin Core Metadata Element Set,” August 2007.)

"nc" NETCONF protocol [1] (Enns, R., “NETCONF Configuration Protocol,” December 2006.)

"en" NETCONF event notifications [10] (Chisholm, S. and H. Trevino, “NETCONF Event Notifications,” July 2008.)

"nma" NETMOD-specific schema annotations (see Section 4.3 (NETMOD-specific Annotations))

"nmt" Conceptual tree (see Section 6.1 (Conceptual Data Tree))

"dsrl" Document Semantics Renaming Language [11] (ISO/IEC, “Information Technology - Document Schema Definition Languages

(DSDL) - Part 8: Document Semantics Renaming Language - DSRL,” 12 2008.)

"rng" RELAX NG [12] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. Second Edition.,” 12 2008.)

"sch" ISO Schematron [13] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron,” 6 2006.)

"xsd" W3C XML Schema [14] (Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, “XML Schema Part 1: Structures Second Edition,” October 2004.)

The following table shows the mapping of these prefixes to namespace URIs.

Prefix	Namespace URI
a	http://relaxng.org/ns/compatibility/annotations/1.0
dc	http://purl.org/dc/terms
nc	urn:ietf:params:xml:ns:netconf:base:1.0
en	urn:ietf:params:xml:ns:netconf:notification:1.0
nma	urn:ietf:params:xml:ns:netmod:dsdl-annotations:1
nmt	urn:ietf:params:xml:ns:netmod:conceptual-tree:1
dsrl	http://purl.oclc.org/dsdl/dsrl
rng	http://relaxng.org/ns/structure/1.0
sch	http://purl.oclc.org/dsdl/schematron
xsd	http://www.w3.org/2001/XMLSchema

Table 1: Used namespace prefixes and corresponding URIs

2. Objectives and Motivation

TOC

The main objective of this work is to complement YANG as a data modeling language by validation capabilities of DSDL schema languages, primarily RELAX NG and Schematron. This document describes the correspondence between grammatical, semantic and data type constraints

expressed in YANG and equivalent DSDL constructs. The ultimate goal is to be able to capture all substantial information contained in YANG modules and express it in DSDL schemas. While the mapping from YANG to DSDL described in this document is in principle invertible, the inverse mapping from DSDL to YANG is not in its scope.
XML-encoded data appear in several different forms in various phases of the NETCONF workflow - configuration datastore contents, RPC requests and replies, and notifications. Moreover, RPC methods are characterized by an inherent diversity resulting from selective availability of capabilities and features. YANG modules can also define new RPC methods. The mapping should be able to accommodate this variability and generate schemas that are specifically tailored to a particular situation and thus considerably more efficient than generic all-encompassing schemas.
In order to cope with this variability, we assume that the schemas can be generated on demand from the available collection of YANG modules and their lifetime will be relatively short. In other words, we don't envision that any collection of DSDL schemas will be created and maintained over extended periods of time in parallel to YANG modules.
The generated schemas are primarily intended as input to the existing XML schema validators and other off-the-shelf tools. However, the schemas may also be perused by developers and users as a formal representation of constraints on a particular XML-encoded data object. Consequently, our secondary goal is to keep the schemas as readable as possible. To this end, the complexity of the mapping is distributed into two steps:

1. The first step maps one or more YANG modules to a single RELAX NG schema of the so-called "conceptual tree", which contains grammatical constraints for the main data tree as well as RPCs and notifications. In order to record additional constraints that appear in the YANG modules but cannot be expressed in RELAX NG, the schema is augmented by simple annotations. The resulting schema should thus be considered a reasonably readable equivalent of the input YANG modules.

2. In the second step, the annotated RELAX NG schema from step 1 is transformed further to a coordinated set of DSDL schemas containing constraints for a particular data object and a specific situation. The DSDL schemas are intended mainly for machine validation using off-the-shelf tools.

3. DSDL Schema Languages

TOC

The mapping described in this document uses three of the DSDL schema languages, namely RELAX NG, Schematron and DSRL.

3.1. RELAX NG

TOC

RELAX NG (pronounced "relaxing") is an XML schema language for grammar-based validation and Part 2 of the ISO/IEC DSDL family of standards [12] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. Second Edition.,” 12 2008.). Like the W3C XML Schema language [14] (Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, “XML Schema Part 1: Structures Second Edition,” October 2004.), it is able to describe constraints on the structure and contents of XML documents. However, unlike the DTD [15] (Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, “Extensible Markup Language (XML) 1.0 (Fourth Edition),” August 2006.) and XSD schema languages, RELAX NG intentionally avoids any infoset augmentation such as defining default values. In the DSDL architecture, the particular task of defining and applying default values is delegated to another schema language, DSRL (see Section 3.3 (Document Semantics Renaming Language (DSRL))).
As its base datatype library, RELAX NG uses the W3C XML Schema Datatype Library [16] (Biron, P. and A. Malhotra, “XML Schema Part 2: Datatypes Second Edition,” October 2004.), but unlike XSD, other datatype libraries may be used along with it or even replace it if necessary.
RELAX NG is very liberal in accepting annotations from other namespaces. With few exceptions, such annotations may be placed anywhere in the schema and need no encapsulating element such as <xsd:annotation> in XSD.
RELAX NG schema can be represented using two equivalent syntaxes: XML and compact. The compact syntax is described in Annex C of the RELAX NG specification [17] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. AMENDMENT 1: Compact Syntax,” 1 2006.), which was added to the standard in 2006 (Amendment 1). Automatic bidirectional conversions between the two syntaxes can be accomplished using for example Trang.
For its terseness and readability, the compact syntax is often the preferred form for publishing RELAX NG schemas whereas validators and other software tools generally require the XML syntax. However, the compact syntax has two drawbacks:

- External annotations make the compact syntax schema considerably less readable. While in the XML syntax the annotating elements and attributes are represented in a simple and uniform way (XML elements and attributes from foreign namespaces), the compact syntax uses four different syntactic constructs: documentation, grammar, initial and following annotations. Therefore, the impact on readability that results from adding annotations is often much stronger for the compact syntax than for the XML syntax.

- In a program, it is more difficult to generate compact syntax than XML syntax. While a number of software libraries exist that make it easy to create an XML tree in memory and serialize it, no such aid is available for compact syntax.

For these reasons, the mapping specification in this document use exclusively the XML syntax. Where appropriate, though, the schemas resulting from the translation may be presented in the equivalent compact syntax.
RELAX NG elements are qualified with the namespace URI "http://relaxng.org/ns/structure/1.0". The namespace of the W3C Schema Datatype Library is "http://www.w3.org/2001/XMLSchema-datatypes".

3.2. Schematron

TOC

Schematron is Part 3 of DSDL that reached the status of a full ISO/IEC standard in 2006 [13] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron,” 6 2006.). In contrast to the traditional schema languages such as DTD, XSD or RELAX NG, which are based on the concept of a formal grammar, Schematron utilizes a rule-based approach. Its rules may specify arbitrary conditions involving data from different parts of an XML document. Each rule consists of three essential parts:

- Context - an XPath expression that defines the set of locations where the rule is to be applied,

- Assert or report condition - another XPath expression that is evaluated relative to the location matched by the context expression.

- Human-readable message that is displayed when the assert condition is false or report condition is true.

The difference between the assert and report condition is that the former is positive in that it states a condition that a valid document has to satisfy, whereas the latter specifies an error condition.
Schematron draws most of its expressive power from XPath [18] (Clark, J. and S. DeRose, “XML Path Language (XPath) Version 1.0,” November 1999.) and XSLT [19] (Clark, J., “XSL Transformations (XSLT) Version 1.0,” November 1999.). ISO Schematron allows for dynamic query language binding so that the following XML query languages can be used: STX, XSLT 1.0, XSLT 1.1, EXSLT, XSLT 2.0, XPath 1.0, XPath 2.0 and XQuery 1.0 (this list may be extended in future).
The human-readable error messages are another feature that distinguishes Schematron from other schema languages such as RELAX NG or XSD. The messages may even contain XPath expressions that are

evaluated in the actual context and thus refer to existing XML document nodes and their content.
ISO Schematron introduced the concept of *abstract patterns* whose purpose is similar to functions in programming languages. The mapping described in this document uses a library of abstract patterns for specifying generic constraints such as uniqueness of certain leaf values in list items.
The rules defined by a Schematron schema may be divided into several subsets known as *phases*. Validations may then be set up to include only selected phases. In the context of NETCONF data validation, this is useful for relaxing constraints that may not always apply. For example, the reference integrity may not be enforced for a candidate configuration.
Schematron elements are qualified with namespace URI "http://purl.oclc.org/dsdl/schematron".

3.3. Document Semantics Renaming Language (DSRL)

TOC

DSRL (pronounced "disrule") is Part 8 of DSDL that reached the status of a full ISO/IEC standard in 2008 [11] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 8: Document Semantics Renaming Language - DSRL,” 12 2008.). Unlike RELAX NG and Schematron, it is specifically designed to modify XML information set of the validated document. The primary application for DSRL is renaming XML elements and attributes. DSRL can also define default values for XML attributes and elements so that elements or attributes with these default values are inserted if they are missing in the validated documents. The latter feature is used by the YANG-to-DSDL mapping for representing YANG defaults for leaf nodes.
DSRL elements are qualified with namespace URI "http://purl.oclc.org/dsdl/dsrl".

---

4. Additional Annotations

TOC

In addition to the DSDL schema languages, the mapping uses three sets of annotations that are added as foreign-namespace elements and attributes to RELAX NG schemas. Two of the annotation sets - Dublin Core elements and DTD compatibility annotations - are standard vocabularies for representing metadata and documentation, respectively. While these data model items may not be used for formal validation, they quite often carry important information. Therefore, they SHOULD be included in the conceptual tree schema and MAY also appear in the final validation schemas.

The third set are NETMOD-specific annotations conveying semantic constraints and other information that cannot be expressed natively in RELAX NG. These annotations are only used in the first step of the mapping, i.e., in the conceptual tree schema. In the second mapping step, these annotations are converted to Schematron and DSRL rules.

4.1. Dublin Core Metadata Elements

Dublin Core is a system of metadata elements that was originally created for describing metadata of World Wide Web resources in order to facilitate their automated lookup. Later it was accepted as a standard for describing metadata of arbitrary resources. This specification uses the definition found in [9] (Kunze, J., “The Dublin Core Metadata Element Set,” August 2007.).
Dublin Core elements are qualified with namespace URI "http://purl.org/dc/terms".

4.2. RELAX NG DTD Compatibility Annotations

DTD compatibility annotations are part of the RELAX NG DTD Compatibility specification [8] (Clark, J., Ed. and M. Murata, Ed., “RELAX NG DTD Compatibility,” December 2001.). The YANG-to-DSDL mapping uses only the <a:documentation> annotation for representing YANG 'description' and 'reference' texts.
Note that there is no intention to make the resulting schemas DTD-compatible, the main reason for using these annotations is technical: they are well supported and adequately interpreted by several RELAX NG tools.
DTD compatibility annotations are qualified with namespace URI "http://relaxng.org/ns/compatibility/annotations/1.0".

4.3. NETMOD-specific Annotations

NETMOD-specific annotations are XML elements and attributes qualified with the namespace URI "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1" that appear in various locations in the conceptual tree schema. YANG statements are mapped to these annotations in a very straightforward way. In particular, the annotation attributes and elements always have the same name as the corresponding YANG statement.
Table 2 (NETMOD-specific annotations) lists alphabetically the names of NETMOD-specific annotation elements (in angle brackets) and attributes

(prefixed with "@") along with a reference to the section where their use is described. Appendix A (RELAX NG Schema for NETMOD-specific Annotations) then contains the RELAX NG schema of this annotation vocabulary.

annotation	section	note
@nma:config	10.9 (The config Statement)	
@nma:default	10.12 (The default Statement)	
@nma:default-case	10.7 (The case Statement)	
<nma:error-app-tag>	10.15 (The error-app-tag Statement)	1
<nma:error-message>	10.16 (The error-message Statement)	1
<nma:instance-identifier>	10.50.6 (The instance-identifier Type)	2
@nma:key	10.23 (The key Statement)	
<nma:leafref>	10.50.7 (The leafref Type)	2
@nma:min-elements	10.25 (The leaf-list Statement)	
@nma:max-elements	10.25 (The leaf-list Statement)	
<nma:must>	10.32 (The must Statement)	3
@nma:ordered-by	10.35 (The ordered-by Statement)	
@nma:status	10.48 (The status Statement)	
@nma:unique	10.52 (The unique Statement)	
@nma:units	10.53 (The units Statement)	
@nma:when	10.56 (The when Statement)	

Table 2: NETMOD-specific annotations

Notes:

1. Appears only as subelement of <nma:must>.

2. Has an optional attribute @require-instance.

3. Has a mandatory attribute @assert and two optional subelements <nma:error-app-tag> and <nma:error-message>.

TOC

5. Overview of the Mapping

This section gives an overview of the YANG-to-DSDL mapping, its inputs and outputs. Figure 1 (Structure of the mapping) presents an overall structure of the mapping:

```
                      +----------------+
                      | YANG module(s) |
                      +----------------+
                              |
                              |T
                              |
              +---------------------------------+
              | DSDL schema for conceptual tree |
              +---------------------------------+
                 /       |           |       \      +-------+
                /        |           |        \     |library|
             Tg/       Tr|           |Tn       \    +-------+
              /          |           |          \
        +---------+   +-----+    +-------+    +------+
        |get reply|   | rpc |    | notif |    | .... |
        +---------+   +-----+    +-------+    +------+
```

Figure 1: Structure of the mapping

The mapping procedure is divided into two steps:

1. Transformation T in the first step maps one or more YANG modules to a single RELAX NG schema for the conceptual tree (see Section 6.1 (Conceptual Data Tree)). Constraints that cannot be expressed directly in RELAX NG (list key definitions, 'must' statements etc.) and various documentation texts are recorded in the schema as simple annotations belonging to special namespaces.

2. In the second step, the conceptual tree schema is transformed in multiple ways to a coordinated set of DSDL schemas that can be used for validating a particular data object in a specific context. Figure 1 (Structure of the mapping) shows just three simplest possibilities as examples. In the process, appropriate parts of the conceptual tree schema are extracted and specific annotations transformed to equivalent, but usually more complex, Schematron patterns, <dsrl:default-content> elements etc.

3. As indicated in Figure 1 (Structure of the mapping), the second step of the mapping also uses a schema-independent library that contains common schema objects such as RELAX NG named pattern definitions.

An implementation of the mapping algorithm accepts one or more valid YANG modules as its input. It is important to be able to process multiple YANG modules together since multiple modules may be negotiated for a NETCONF session and the contents of the configuration datastore is then obtained as the union of data trees specified by the individual modules, which may also lead to multiple roots. In addition, the input modules may be further coupled by the 'augment' statement in which one module augments the data tree of another module.
It is also assumed that the algorithm has access, perhaps on demand, to all YANG modules that the module(s) imports (transitively).
The output of the first mapping step is an annotated RELAX NG schema for the conceptual tree, which represents a virtual XML document containing, in its different subtrees, the entire datastore, all RPC requests and replies, and notifications defined by the input YANG modules. By "virtual" we mean that such an XML document need not correspond to the actual layout of the configuration database in a NETCONF agent, and will certainly never appear on the wire as the content of a NETCONF PDU. Hence, the RELAX NG schema for the conceptual tree is not intended for any direct validations but rather as a representation of a particular data model expressed in RELAX NG and the common starting point for subsequent transformations that will typically produce validation schemas. The conceptual tree is further described in Section 6.1 (Conceptual Data Tree).
Other information contained in input YANG modules, such as semantic constraints or default values, are recorded as annotations - XML elements or attributes qualified with namespace URI "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1". Metadata describing the YANG modules are mapped to annotations utilizing Dublin Core elements (Section 4.1 (Dublin Core Metadata Elements)). Finally, documentation strings are mapped to the <a:documentation> elements belonging to the DTD compatibility vocabulary (Section 4.2 (RELAX NG DTD Compatibility Annotations)).
The output from the second step is is a coordinated set of three DSDL schemas corresponding to a specific data object and context:

*RELAX NG schema describing the grammatical and datatype constraints;

*Schematron schema expressing other constraints such as uniqueness of list keys or user-specified semantic rules;

*DSRL schema containing a specification of default values.

6. Design Considerations

TOC

YANG modules could be mapped to DSDL schemas in a number of ways. The mapping procedure described in this document uses several specific design decisions that are discussed in the following subsections.

6.1. Conceptual Data Tree

TOC

DSDL schemas generated from YANG modules using the procedure described in this document are intended to be used for validating XML-encoded NETCONF data in various forms (full datastore and several types of PDUs): every YANG-based model represents the contents of a full datastore but also implies an array of schemas for all types of NETCONF PDUs. For a reasonably strict validation of a given data object, the schemas have to be quite specific. To begin with, effective validation of NETCONF PDU content requires separation of client and server schemas. While the decision about proper structuring of all PDU-validating schemas is beyond the scope of this document, the mapping procedure is designed to accommodate any foreseeable validation needs.
An essential part of the YANG-to-DSDL mapping procedure is nonetheless common to all validation approaches: the grammar and datatype specifications for the datastore, RPCs and notifications expressed by one or more YANG modules have to be translated to RELAX NG. In order to be able to separate this common step, we introduce the notion of *conceptual data tree*: it is a virtual XML tree that contains full datastore, RPC requests with corresponding replies and notifications. The schema for the conceptual tree - a single RELAX NG schema with annotations - therefore has a quite similar logic as the input YANG module(s), the only major difference being the RELAX NG schema language.
The conceptual data tree may be schematically represented as follows:

```
     <nmt:netmod-tree
         xmlns:nmt="urn:ietf:params:xml:ns:netmod:conceptual-tree:1">
       <nmt:top>
         ... configuration and status data ...
       </nmt:top>
       <nmt:rpc-methods>
         <nmt:rpc-method>
           <nmt:input>
             <myrpc ...>
               ...
             </myrpc>
           </nmt:input>
           <nmt:output>
             ...
           </nmt:output>
         </nmt:rpc-method>
         ...
       </nmt:rpcs>
       <nmt:notifications>
         <nmt:notification>
           <mynotif>
             ...
           </mynotif>
         </nmt:notification>
         ...
       </nmt:notifications>
     </nmt:netmod>
```

The namespace URI "urn:ietf:params:xml:ns:netmod:conceptual-tree:1" identifies a simple vocabulary consisting of a few elements that encapsulate and separate the various parts of the conceptual data tree. The conceptual tree schema is not intended for direct validation but rather serves as a well-defined starting point for subsequent transformations that generate various validation schemas. Such transformations should be relatively simple, they will typically extract one or several subtrees from the conceptual tree schema, modify them as necessary and add encapsulating elements such as those from the NETCONF RPC layer.

Additional information contained in the source YANG module(s), such as semantic constraints and default values, is represented in the form of interim NETMOD-specific annotations that are included as foreign-namespace elements or attributes in the RELAX NG schema for the conceptual tree. In the second phase of the mapping, these annotations are translated to equivalent Schematron and DSRL rules.

As a useful side effect, by introducing the conceptual data tree we are also able to resolve the difficulties arising from the fact that a single configuration repository may consist of multiple parallel data hierarchies defined in one or more YANG modules, which cannot be mapped to a valid XML document. In the conceptual data tree, such multiple hierarchies appear under the single <nmt:top> element.

6.2. Modularity

Both YANG and RELAX NG offer means for modularity, i.e., for splitting the contents into separate modules (schemas) and combining or reusing them in various ways. However, the approaches taken by YANG and RELAX NG differ. Modularity in RELAX NG is suitable for ad hoc combinations of a small number of schemas whereas YANG assumes a large set of modules similar to SNMP MIBs. The following differences are important:

* In YANG, whenever module A imports module B, it gets access to the definitions (groupings and typedefs) appearing at the top level of module B. However, no part of module B's data tree is imported along with it. In contrast, the <rng:include> pattern in RELAX NG imports both definitions of named patterns and the entire schema tree from the included schema.

* The names of imported YANG groupings and typedefs are qualified with the namespace of the imported module. On the other hand, the data nodes contained inside the imported groupings, when used within the importing module, become part of the importing namespace. In RELAX NG, the names of patterns are unqualified and so named patterns defined in both the importing and imported module share the same flat namespace. The contents of RELAX NG named patterns may either keep the namespace of the schema where they are defined or inherit the namespace of the importing module, analogically to YANG. However, in order to achieve the latter behavior, the imported module must be prepared in a special way as a library module that cannot be used as a stand-alone schema.

So the conclusion is that the modularity mechanisms of YANG and RELAX NG, albeit similar, are not directly compatible. Therefore, the corresponding design decision for the mapping algorithm is to collect all information in a single schema for the conceptual tree, even if it comes from multiple YANG modules or submodules. In other words, translations of imported groupings and typedefs are installed in the translation of importing module - but only if they are really used there.
NOTE: The 'include' statement that is used in YANG for including submodules has in fact the same semantics as the <rng:include> pattern. However, since we don't map the modular structure for YANG modules, it seems to have little sense to do it for submodules. Consequently, the contents of submodules appear directly in the conceptual tree schema, too.

6.3. Granularity

TOC

RELAX NG supports different styles of schema structuring: One extreme, often called "Russian Doll", specifies the structure of an XML instance document in a single hierarchy. The other extreme, the flat style, uses a similar approach as the Data Type Definition (DTD) schema language - every XML element is introduced inside a new named pattern. In practice, some compromise between the two extremes is usually chosen. YANG supports both styles in principle, too, but in most cases the modules are organized in a way that's closer to the "Russian Doll" style, which provides a better insight into the structure of the configuration data. Groupings are usually defined only for contents that are prepared for reuse in multiple places via the 'uses' statement. In contrast, RELAX NG schemas tend to be much flatter, because finer granularity is also needed in RELAX NG for extensibility of the schemas - it is only possible to replace or modify schema fragments that are factored out as named patterns. For YANG this is not an issue since its 'augment' and 'refine' statements can delve, by using path expressions, into arbitrary depths of existing structures. In general, it not feasible to map YANG extension mechanisms to those of RELAX NG. For this reason, the mapping essentially keeps the granularity of the original YANG data model: definitions of named patterns in the resulting RELAX NG schema usually have direct counterparts in YANG groupings and definitions of derived types.

6.4. Handling of XML Namespaces

TOC

Most modern XML schema languages including RELAX NG, Schematron and DSRL support schemas for so-called compound XML documents, which contain elements from multiple namespaces. This is useful for our purpose since the YANG-to-DSDL mapping algorithm allows for multiple input YANG modules that naturally leads to compound document schemas. RELAX NG offers two alternatives for defining the "target" namespaces in the schema:

1. First possibility is the traditional XML way via the @xmlns:xxx attribute.

2. One of the target namespace URIs may be declared using the @ns attribute.

In both cases these attributes are typically attached to the <rng:grammar> element.

The design decision for the mapping is to use exclusively the alternative 1, since all YANG modules are represented symmetrically, which makes further processing of the conceptual tree schema considerably easier. Moreover, this way the namespace prefixes declared in all input modules are recorded in the resulting schema - the prefix for the namespace declared using @ns would be lost.
In contrast, there is no choice for Schematron and DSRL since both schema languages require the target namespaces to be defined by special means. In Schematron, <sch:ns> subelements of the root <sch:schema> element serve this purpose, whereas in DSRL it is the @targetNamespace attribute of the root <dsrl:maps> element.

7. Mapping YANG Data Models to the Conceptual Tree Schema

TOC

This section explains the main principles underlying the first step of the mapping. Its result is an annotated RELAX NG schema of the conceptual tree, which is described in Section 6.1 (Conceptual Data Tree).
As a special case, if the input YANG modules contain no data nodes (this is typical e.g., for datatype library modules), an implementation MAY entirely remove the schema of the (empty) conceptual tree - the <rng:start> element with all its contents. The output RELAX NG schema will then contain only named pattern definitions translated from YANG groupings and typedefs.
Detailed specification of the mapping of individual YANG statements is contained in Section 10 (Mapping YANG Statements to Annotated RELAX NG).

7.1. Optional and Mandatory Content

TOC

In YANG, the decision whether a given data node is mandatory or optional is driven by the following rules, see [5] (Bjorklund, M., Ed., “YANG - A data modeling language for NETCONF,” March 2009.), Section 3.1:
Leaf and choice nodes are mandatory if they contain the substatement

```
     mandatory true;
```

In addition, leaf nodes are mandatory if they are declared as list keys.
Lists or leaf-lists are mandatory if they contain 'min-elements' substatement with argument value greater than zero.
A slightly more complicated situation arises for YANG containers.
First, containers with the 'presence' substatement are always optional

since their presence or absence carries specific information. On the other hand, non-presence containers may be omitted if they are empty. This leads to the following recursive rule:
A container node is optional if its definition contains the 'presence' substatement or none of its child nodes is mandatory.
In RELAX NG, all elements that are optional must be explicitly wrapped in the <rng:optional> element. The mapping algorithm thus uses the above rules to determine whether a YANG node is optional and if so, insert the <rng:optional> element in the RELAX NG schema.

7.2. Mapping YANG Groupings and Typedefs

TOC

YANG groupings and typedefs are generally mapped to RELAX NG named patterns. There are, however, several caveats that the mapping has to take into account.
First of all, YANG typedefs and groupings may appear at all levels of the module hierarchy and are subject to lexical scoping, see [5] (Bjorklund, M., Ed., “YANG - A data modeling language for NETCONF,” March 2009.), Section 5.5. Moreover, top-level symbols from external modules are imported as qualified names represented using the external module namespace prefix and the name of the symbol. In contrast, named patterns in RELAX NG (both local and imported via the <rng:include> pattern) share the same namespace and within a grammar they are always global - their definitions may only appear at the top level as children of the <rng:grammar> element. Consequently, whenever YANG groupings and typedefs are mapped to RELAX NG named pattern definitions, their names MUST be disambiguated in order to avoid naming conflicts. The mapping uses the following procedure for mangling the names of groupings and type definitions:

- Names of groupings and typedefs appearing at the *top level* of the YANG module hierarchy are prefixed with the module name and two underscore characters ("__").

- Names of other groupings and typedefs, i.e., those that do not appear at the top level of a YANG module, are prefixed with the module name, double underscore, and then the names of all ancestor data nodes separated by double underscore.

- Since the names of groupings and typedefs in YANG have different namespaces, an additional underline character is added to the front of the mangled names of all groupings.

For example, consider the following YANG module which imports the standard module "inet-types" [20] (Schoenwaelder, J., Ed., “Common YANG Data Types,” November 2008.):

```
     module example1 {
         namespace "http://example.com/ns/example1";
         prefix "ex1";
         import "inet-types" {
             prefix "inet";
         }
         typedef vowels {
             type string {
                 pattern "[aeiouy]*";
             }
         }
         grouping "grp1" {
             leaf "void" {
                 type "empty";
             }
         }
         container "cont" {
             grouping "grp2" {
                 leaf "address" {
                     type "inet:ip-address";
                 }
             }
             leaf foo {
                 type vowels;
             }
             uses "grp1";
             uses "grp2";
         }
     }
```

The resulting RELAX NG schema will then contain the following named pattern definitions (long regular expression patterns for IPv4 and IPv6 addresses are not shown):

```
     <rng:define name="example1__vowels">
       <rng:data type="string">
         <rng:param name="pattern">[aeiouy]*</param>
       </rng:data>
     </rng:define>

     <rng:define name="_example1__grp1">
       <rng:optional>
         <rng:element name="t:void">
           <rng:empty/>
         </rng:element>
       </rng:optional>
     </rng:define>

     <rng:define name="_example1__cont__grp2">
       <rng:optional>
         <rng:element name="t:address">
           <rng:ref name="inet-types__ip-address"/>
         </rng:element>
       </rng:optional>
     </rng:define>

     <rng:define name="inet-types__ip-address">
       <rng:choice>
         <rng:ref name="inet-types__ipv4-address"/>
         <rng:ref name="inet-types__ipv6-address"/>
       </rng:choice>
     </rng:define>

     <rng:define name="inet-types__ipv4-address">
       <rng:data type="string">
         <rng:param name="pattern">... regex pattern ...</param>
       </rng:data>
     </rng:define>

     <rng:define name="inet-types__ipv6-address">
       <rng:data type="string">
         <rng:param name="pattern">... regex pattern ...</param>
       </rng:data>
     </rng:define>
```

7.2.1. YANG Refinements and Augments

YANG groupings represent a similar concept as named pattern definitions in RELAX NG and both languages also offer mechanisms for their subsequent modification. However, in RELAX NG the definitions themselves are modified whereas YANG allows for modifying *expansions* of groupings. Specifically, YANG provides two statements for this purpose that may appear as substatements of 'uses':

* 'refine' statement allows for changing parameters of a schema node inside the grouping referenced by the parent 'uses' statement;

* 'augment' statement can be used for adding new schema nodes to the grouping content.

Both 'refine' and 'augment' statements are quite powerful in that they can address, using a subset of XPath 1.0 expressions as arguments, schema nodes that are arbitrarily deep inside the grouping content. In contrast, definitions of named patterns in RELAX NG operate exclusively at the topmost level of the named pattern content. In order to achieve a modifiability of named patterns comparable to YANG, the RELAX NG schema would have to be extremely flat (cf. Section 6.3 (Granularity)) and very difficult to read.

Since the goal of the mapping described in this document is to generate ad hoc DSDL schemas, we decided to avoid these complications and instead expand the grouping and refine and/or augment it "in place". In other words, every 'uses' statement which has 'refine' and/or 'augment' substatements is virtually replaced by the content of the corresponding grouping, the changes specified in the 'refine' and 'augment' statements are applied and the resulting YANG schema fragment is mapped as if the 'uses'/'grouping' indirection wasn't there.

If there are further 'uses' statements inside the grouping content, they may require expansion, too: it is necessary if the contained 'uses'/'grouping' pair lies on the "modification path" specified in the argument of a 'refine' or 'augment' statement.

EXAMPLE. Consider the following YANG module:

```
     module example2 {
         namespace "http://example.com/ns/example2";
         prefix ex2;
         grouping leaves {
             uses fr;
             uses es;
         }
         grouping fr {
             leaf feuille {
                 type string;
             }
         }
         grouping es {
             leaf hoja {
                 type string;
             }
         }
         uses leaves;
     }
```

The resulting conceptual tree schema contains three named pattern definitions corresponding to the three groupings, namely

```
     <rng:define name="_example2__leaves">
       <rng:ref name="_example2__fr"/>
       <rng:ref name="_example2__es"/>
     </rng:define>

     <rng:define name="_example2__fr">
       <rng:optional>
         <rng:element name="feuille">
           <rng:data type="string"/>
         </rng:element>
       </rng:optional>
     </rng:define>

     <rng:define name="_example2__es">
       <rng:optional>
         <rng:element name="hoja">
           <rng:data type="string"/>
         </rng:element>
       </rng:optional>
     </rng:define>
```

and the configuration data part of the conceptual tree schema is a single named pattern reference:

```
     <rng:ref name="_example2__leaves"/>
```

Now assume that the "uses leaves" statement is refined:

```
     uses leaves {
         refine "hoja" {
             default "alamo";
         }
     }
```

The resulting conceptual tree schema now contains just one named pattern definition - "_example__fr". The other two groupings "leaves" and "es" have to be expanded because they both lie on the "modification path", i.e., contain the leaf "hoja" that is being refined. The configuration data part of the conceptual tree now looks like this:

```
     <rng:ref name="_example2__fr"/>
     <rng:optional>
       <rng:element name="hoja" nma:default="alamo">
         <rng:data type="string"/>
       </rng:element>
     </rng:optional>
```

7.2.2. Type derivation chains

TOC

RELAX NG has no equivalent of the type derivation mechanism in YANG, where a base built-in type may be modified (in multiple steps) by adding new restrictions. Therefore, when mapping YANG derived types with restrictions, the derived types MUST be "unwound" all the way back to the base built-in type. At the same time, all restrictions found along the type derivation chain MUST be combined and their intersection used as facets restricting the corresponding type in RELAX NG.
When a derived YANG type is used without restrictions, the 'type' statement is mapped simply to the <rng:ref> element, i.e., a named pattern reference. However, if restrictions are specified as substatements of the 'type' statement, the type MUST be expanded at that point so that only the base built-in type appears in the output schema, restricted with facets that again correspond to the combination of all restrictions found along the type derivation chain and also in the 'type' statement.
EXAMPLE. Consider this YANG module:

```
     module example3 {
         namespace "http://example.com/ns/example3";
         prefix ex3;
         typedef dozen {
             type uint8 {
                 range 1..12;
             }
         }
         leaf month {
             type dozen;
         }
```

The 'type' statement in "leaf month" is mapped simply to the reference <rng:ref name="example__dozen"/> and the corresponding named pattern is defined as follows:

```
     <rng:define name="example3__dozen">
       <rng:data type="unsignedByte">
         <rng:param name="minInclusive">1</param>
         <rng:param name="maxInclusive">12</param>
       </rng:data>
     </rng:define>
```

Assume now that the definition of leaf "month" is changed to

```
     leaf month {
         type dozen {
             range 7..max;
         }
     }
```

The output RELAX NG schema then won't contain any named pattern definition and leaf "month" will be mapped directly to

```
     <rng:element name="month">
       <rng:data type="unsignedByte">
         <rng:param name="minInclusive">7</param>
         <rng:param name="maxInclusive">12</param>
       </rng:data>
     </rng:element>
```

7.3. Translation of XPath Expressions

TOC

YANG uses full XPath 1.0 syntax [18] (Clark, J. and S. DeRose, “XML Path Language (XPath) Version 1.0,” November 1999.) for the arguments

of 'must' and 'when' statements and a subset thereof in several other statements. However, since the name of a data node always belongs to the namespace of the YANG Module where the data node is defined, YANG adopted a simplification similar to the concept of *default namespace* in XPath 2.0: node names needn't carry a namespace prefix inside the module where they are defined, in which case the module's namespace is assumed.
If an XPath expression is carried over to a NETMOD-specific annotation in the first mapping step, it MUST be translated into a fully conformant XPath 1.0 expression that also reflects the hierarchy of the conceptual data tree:

1. Each unprefixed node name MUST be prepended with the local module's namespace prefix declared by the 'prefix' statement.

2. Absolute XPath expressions, i.e., those starting with a slash, MUST be prepended with appropriate path in the conceptual tree, according to the YANG specification of context for XPath expressions, see [18] (Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0," November 1999.), sections 7.5.3 and 7.19.5.

Translation rule 2 means for example that absolute XPath expressions appearing in the main configuration data tree always start with "nmt:netmod-tree/nmt:top/", those appearing in a notification always start with "nmt:netmod-tree/nmt:notifications/nmt:notification/", etc.
EXAMPLE. YANG XPath expression "/dhcp/max-lease-time" appearing in the main configuration data will be translated to "nmt:netmod-tree/nmt:top/dhcp:dhcp/dhcp:max-lease-time".
[Editor's note: We may want to introduce "$root" variable that always contains the appropriate partial path in conceptual tree. The translated XPath in the example would then become "$root/dhcp:dhcp/dhcp:max-lease-time".]
The key identifiers and "descendant schema node identifiers" (see the ABNF production for "descendant-schema-nodeid" in Section 12 of [5] (Bjorklund, M., Ed., "YANG - A data modeling language for NETCONF," March 2009.)) that appear as items in the arguments of 'key' and 'unique' statements, respectively, are special XPath expressions and MUST be translated in the same way, i.e., after the translation each key and every component of a node identifier must have the namespace prefix of the local module.

7.4. YANG Language Extensions

TOC

YANG allows for extending its own language in-line by adding new statements with keywords from special namespaces. Such extensions first have to be declared using the 'extension' statement and then can be used as the native statements, only with a namespace prefix qualifying the extension keyword. RELAX NG has a similar extension mechanism - XML elements and attributes with names from foreign namespaces may be inserted at almost every place of a RELAX NG schema.

YANG language extensions may or may not have a meaning in the context of DSDL schemas. Therefore, an implementation MAY ignore any or all of the extensions. However, an extension that is not ignored MUST be mapped to XML element(s) and/or attribute(s) that exactly match the YIN form of the extension.

EXAMPLE. Consider the following extension defined by the "acme" module:

```
extension documentation-flag {
    argument number;
}
```

This extension can then be used in the same or another module, for instance like this:

```
leaf folio {
    acme:documentation-flag 42;
    type string;
}
```

If this extension is honored by the mapping, it will be mapped to

```
<rng:element name="folio">
   <acme:documentation-flag number="42"/>
   <rng:data type="string"/>
</rng:element>
```

Note that the 'extension' statement itself is not mapped in any way.

8. Mapping Conceptual Tree Schema to DSDL

TOC

As explained in Section 5 (Overview of the Mapping), the second step of the YANG-to-DSDL mapping takes the conceptual tree schema and transforms it to various DSDL schemas ready for validation. As an input parameter, this step gets in the simplest case a specification of the NETCONF XML document type (or combination of multiple types) that is to be validated. These document type can be for example reply to <nc:get> or <nc:get-config>, RPC requests or replies and notification. Other parameters further describing the context may also be provided, such as the list of active capabilities, features etc.

In general, the second mapping step has to accomplish the following three tasks:

1. Extract the part(s) of the conceptual tree schema that are appropriate or the requested document type. For example, if a <get> reply is to be validated, the subtree under <nmt:top> must be selected.

2. The schema must be accommodated to the specific encapsulating XML elements mandated by the RPC layer. These are, for example, <nc:rpc> and <nc:data> elements in the case of a datastore or <en:notification> for a notification.

3. Finally, NETMOD-specific annotations that are relevant for the schema language of the generated schema must be mapped to corresponding schema-language-specific rules.

These three tasks are together much simpler than the first mapping step. Presumably, they can be effectively realized using XSL transformations [19] (Clark, J., "XSL Transformations (XSLT) Version 1.0," November 1999.).
The following subsections describe the details of the second mapping step for the individual DSDL schema languages. Section 11 (Mapping NETMOD-specific annotations to DSDL Schema Languages) then contains a detailed specification for the mapping of all NETMOD-specific annotations.

8.1. Generating RELAX NG Schemas for Various Document Types

With one minor exception, obtaining a validating RELAX NG schema from the conceptual tree schema really means only taking appropriate parts from the conceptual tree schema and assembling them in a new RELAX NG grammar, perhaps after removing all unwanted annotations. Depending on the XML document type that is the target for validation (<get>/<get-config> reply, RPC or notification) a corresponding top-level part of the grammar MUST be added as described in the following subsections. Schemas for multiple alternative target document types can also be easily generated by enclosing the definitions for requested type in <rng:choice> element.
In order to avoid copying identical named pattern definitions to the output RELAX NG file, these schema-independent definition are collected in a library file "relang-lib.rng" which is then included by the validating RELAX NG schemas. Appendix B (Schema-Independent Library) has the listing of this library file.
The minor exception mentioned above is the annotation @nma:config, which must be observed if the target document type is <get-config> reply. In this case, each element definition that has this attribute

with the value "false" MUST be removed from the schema together with its descendants. See Section 11.1 (The @nma:config Annotation) for more details.

8.1.1. Reply to <get> or <get-config>

TOC

For a reply to <get> or <get-config>, the mapping must take the part of the conceptual tree schema under the definition of <nmt:top> and insert it in the following grammar:

```
<rng:grammar ... namespaces etc. ...>
  <rng:include href="relaxng-lib.rng"/>
  <rng:start>
    <rng:element name="nc:rpc-reply">
      <rng:ref name="message-id-attribute"/>
      <rng:element name="nc:data">
        ... patterns defining contents of "nmt:top" subtree ...
      </rng:element>
    </rng:element>
  </rng:start>
  ... named pattern definitions ...
</rng:grammar>
```

The definition for the named pattern "message-id-attribute" is found in the library file "relaxng-lib.rng" which is included on the second line (see Appendix B (Schema-Independent Library)).
Definitions of other named patterns MUST be copied from the conceptual tree schema without any changes to the resulting grammar. However, an implementation MAY choose to copy only those definitions that are really used in the particular output grammar.

8.1.2. Remote Procedure Calls

TOC

For an RPC method named "myrpc" and defined in a YANG module with prefix "yam", the corresponding schema subtree is identified by the definition of <nmt:rpc-method> element whose <nmt:input> subelement has <yam:myrpc> as the only child.
The mapping must also take into account whether the target document type in an RPC request or reply. For "yam:myrpc" request, the resulting grammar looks as follows:

```
     <rng:grammar ... namespaces etc. ...>
       <rng:include href="relaxng-lib.rng"/>
       <rng:start>
         <rng:element name="nc:rpc">
           <rng:ref name="message-id-attribute"/>
           <rng:element name="yam:myrpc">
             ... patterns defining contents of subtree ...
             ... "nmt:rpc-method/nmt:input/yam:myrpc" ...
           </rng:element>
         </rng:element>
       </rng:start>
       ... named pattern definitions ...
     </rng:grammar>
```

For "myrpc" reply, the output grammar is

```
     <rng:grammar ... namespaces etc. ...>
       <rng:include href="relaxng-lib.rng"/>
       <rng:start>
         <rng:element name="nc:rpc-reply">
           <rng:ref name="message-id-attribute"/>
           ... patterns defining contents of corresponding ...
           ... "nmt:rpc-method/nmt:output" subtree ...
         </rng:element>
       </rng:start>
       ... named pattern definitions ...
     </rng:grammar>
```

In both cases, exact copies of named pattern definitions from the conceptual tree schema MUST be inserted, but an implementation MAY choose to include only those used for the given RPC.

8.1.3. Notifications

TOC

For a notification named "mynotif" and defined in a YANG module with prefix "yam", the corresponding schema subtree is identified by the definition of <nmt:notification> element that has the single child <yam:mynotif>.
The resulting grammar looks as follows:

```
     <rng:grammar ... namespaces etc. ...>
       <rng:include href="relaxng-lib.rng"/>
       <rng:start>
         <rng:element name="en:notification">
           <rng:ref name="eventTime-element"/>
           <rng:element name="yam:myrpc">
             <!-- patterns defining contents of
                  "nmt:rpc-notification/yam:mynotif" subtree -->
           </rng:element>
         </rng:element>
       </rng:start>
       <!-- named pattern definitions -->
     </rng:grammar>
```

The definition of the named pattern "eventTime-element" is found in the "relaxng-lib.rng" library file.
And again, exact copies of named pattern definitions from the conceptual tree schema MUST be inserted, but an implementation MAY choose to include only those used for the given notification.

8.2. Mapping Semantic Constraints to Schematron

TOC

Schematron schemas tend to be much flatter and more uniform compared to RELAX with exactly four levels of XML hierarchy: <sch:schema>, <sch:pattern>, <sch:rule> and <sch:assert> or <sch:report>.
In a Schematron schema generated by the second mapping step, the basic unit of organization is a *rule* represented by the <sch:rule> element. Every rule corresponds to exactly one element definition in the conceptual tree schema. The mandatory @context attribute of <sch:rule> is set to the absolute path of the corresponding element in the data tree.
In the opposite direction, however, not every element definition has a corresponding rule in the Schematron schema: only those definitions are taken into account that are annotated with at least one of the following NETMOD-specific annotations: <nma:instance-identifier>, @nma:key, <nma:leafref>, @nma:min-elements, @nma:max-elements, <nma:must>, @nma:unique and <nma:when>.
Schematron rules may be further grouped into *patterns* represented by the <sch:pattern> element. The mapping uses patterns only for discriminating between subsets of rules that belong to different validation phases, see Section 8.2.1 (Validation Phases). Therefore, the <sch:schema> always has exactly two <sch:pattern> children: one named "standard" contains rules for all annotations except <nma:instance-identifier> and <nma:leafref>, and another named "ref-

integrity" containing rules for these two remaining annotations, i.e., referential integrity checks.
Element definitions in the conceptual tree schema that appear inside a named pattern definition (i.e., have <rng:define> among its ancestors) are subject to a different treatment. This is because their path in the data tree is not fixed - the named pattern may be referred to in multiple different places. The mapping uses *abstract rules* to handle this case: An element definition inside a named pattern is mapped to an abstract rule and every use of the named pattern then extends this abstract pattern in the concrete context.
EXAMPLE. Consider this element definition annotated with <nma:must>:

```
     <rng:element name="dhcp:default-lease-time">
       <rng:data type="unsignedInt"/>
       <nma:must assert=". &lt;= ../dhcp:max-lease-time">
         <nma:error-message>
           The default-lease-time must be less than max-lease-time
         </nma:error-message>
       </nma:must>
     </rng:element>
```

If this element definition appears outside any named pattern and as a child of <dhcp:dhcp> (as it does in the DHCP schema, see Appendix C.2 (Conceptual Tree Schema)), it is mapped to the following Schematron rule:

```
     <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                        dhcp:default-lease-time">
       <sch:assert test=". &lt;= ../dhcp:max-lease-time">
         The default-lease-time must be less than max-lease-time
       </sch:assert>
     </sch:rule>
```

Now assume the element definition is inside a named pattern definition, say

```
     <rng:define name="_dhcp__default-lease-time">
       <rng:element name="dhcp:default-lease-time">
         ... same content ...
       </rng:element>
     </rng:define>
```

In this case it is mapped to an abstract rule:

```
     <sch:rule id="id31415926" abstract="true">
       <sch:assert test=". &lt;= ../dhcp:max-lease-time">
         The default-lease-time must be less than max-lease-time
       </sch:assert>
     </sch:rule>
```

Any use of the named pattern definition via <rng:ref name="_dhcp__default-lease-time"/> then results in a new rule extending the abstract one, for example

```
     <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:default-lease-time">
       <sch:extends rule="id31415926"/>
     </sch:rule>
```

Care must be taken that the value of the @context attribute in general consists of two parts in this case: its beginning is determined by the location of the <rng:ref> element in the main schema tree and the rest of the path comes from the relative position of the annotated element definition inside the named pattern. The situation becomes even more complex when the mapping has to deal with chained definitions of named patterns (<rng:ref> inside <rng:define>). The @context value then must be recursively glued together from multiple parts.
The mapping from the conceptual tree schema to Schematron proceeds in the following steps:

1. First, the active subtree(s) of the conceptual tree schema must be selected depending on the requested target document type. This procedure is identical to the RELAX NG case, including the handling of @nma:config if the target document type is <get-config> reply.

2. Namespaces of all input YANG modules, together with the namespaces of base NETCONF ("nc" prefix) or notifications ("en" prefix) MUST be declared using the <sch:ns> element, for example

   ```
           <sch:ns uri="http://example.com/ns/dhcp" prefix="dhcp"/>
   ```

3. Validation phases are defined (see Section 8.2.1 (Validation Phases)) and their constituting patterns "standard" and "ref-integrity" created.

4. For either validation phase, the input conceptual tree schema is scanned and element definitions with annotations relevant for the given phase are selected and a <sch:rule> is created for each of them. The rule is abstract if the element definition appears inside a named pattern, see above.

5. All annotations attached to the given element definition are then mapped using the mapping rules specified in Section 11 (Mapping NETMOD-specific annotations to DSDL Schema Languages). The resulting <sch:assert> or <sch:report> elements are the installed as children of the <sch:rule> element.

8.2.1. Validation Phases

In certain situations it is useful to validate XML instance documents without enforcing the referential integrity constraints represented by the <nma:leafref> and <nma:instance-identifier> annotations. For example, a candidate configuration refering to configuration parameters or state data of certain hardware will not pass full validation before the hardware is installed. To handle this, the Schematron mapping introduces two *validation phases*:

*Validation phase "full", which is the default, checks all semantic constraints.

*Validation phase "noref" is the same as "full" except it doesn't check referential integrity constraints.

A parameter identifying the validation phase to use has to be passed to the Schematron processor or otherwise both patterns are used by default. How this is exactly done depends on the concrete Schematron processor and is outside the scope of this document.
The validation phases are defined in Schematron by listing the patterns that are to be applied for each phase. Therefore, the mapping puts the rules for referential integrity checking to a special <sch:pattern> with @id attribute set to "ref-integrity". The rules mapped from the remaining semantic constraints are put to another <sch:pattern> with @id attributes set to "standard".
With validation phases, the resulting Schematron schema has the following overall structure:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:ns uri="..." prefix="..."/>
  ... more NS declarations ...
  <sch:phase id="full">
    <sch:active pattern="standard"/>
    <sch:active pattern="ref-integrity"/>
  </sch:phase>
  <sch:phase id="noref">
    <sch:active pattern="standard"/>
  </sch:phase>
  <sch:pattern id="standard">
    ... all rules except ref. integrity checks ...
  </sch:pattern>
  <sch:pattern id="ref-integrity">
    ... rules for ref. integrity checks ...
  </sch:pattern>
</sch:schema>
```

8.3. Mapping Default Values to DSRL

TOC

TBD

9. NETCONF Content Validation

TOC

This section describes the procedures for validating XML instance documents corresponding to various NETCONF PDUs given the set of DSDL schemas generated for the particular document type.
[Editor's note: This section is incomplete. We have to figure out what are the NETCONF instances we want to validate, and also the validation contexts and modes. However, these questions are not DSDL-specific and should be addressed by the WG.]
The validation proceeds in the following steps, see also Figure 2 (Outline of the validation procedure):

1. The XML instance document can be immediately checked for grammatical and data type validity using the RELAX NG schema.

2. Second, the default values for leaves and default cases have to be applied. It is important to apply the defaults before the next validation step because [5] (Bjorklund, M., Ed., "YANG - A data modeling language for NETCONF," March 2009.) states that the data tree against which XPath expressions are evaluated

already has all defaults filled-in. Note that this step modifies the information set of the input XML document.

3. The semantic constraints are checked using the Schematron schema.

```
+----------+                        +----------+
|          |                        |   XML    |
|   XML    |                        | document |
| document |-----------o----------->|   with   |
|          |           ^            | defaults |
|          |           |            |          |
+----------+           |            +----------+
     ^                 | filling-in       ^
     | grammar,        | defaults         | semantic
     | datatypes       |                  | constraints
     |                 |                  |
+----------+       +--------+       +------------+
| RELAX NG |       |  DSRL  |       | Schematron |
|  schema  |       | schema |       |   schema   |
+----------+       +--------+       +------------+
```

Figure 2: Outline of the validation procedure

10. Mapping YANG Statements to Annotated RELAX NG

TOC

Each subsection in this section is devoted to one YANG statement and provides the specification how the statement is mapped to the annotated RELAX NG schema of the conceptual tree. This is the first step of the mapping procedure, see Section 5 (Overview of the Mapping). The subsections are sorted alphabetically by the statement keyword.
Each YANG statement is mapped to an XML fragment, typically a single element or attribute but it may also be a larger structure. The mapping algorithm is inherently recursive, which means that after finishing a statement the mapping continues with its substatements, if there are any, and a certain element of the resulting fragment becomes the parent

of other fragments resulting from the mapping of substatements. We use the following notation:

- *The argument of the statement being mapped is denoted by ARGUMENT.
- *The element in the RELAX NG schema that becomes the parent of the resulting XML fragment is denoted by PARENT.

10.1. The anyxml Statement

TOC

This statement is mapped to <rng:element> element and ARGUMENT becomes the value of its @name attribute. The content of <rng:element> is

```
<rng:ref name="__anyxml__"/>
```

Substatements of the 'anyxml' statement are mapped to additional children of the RELAX NG element definition.
If the 'anyxml' statement occurs in any of the input YANG modules, the following pattern definition MUST be added exactly once to the RELAX NG schema as a child of the <rng:grammar> element (cf. [21] (van der Vlist, E., “RELAX NG,” 2004.), p. 172):

```
<rng:define name="__anyxml__">
  <rng:zeroOrMore>
    <rng:choice>
      <rng:attribute>
        <rng:anyName/>
      </rng:attribute>
      <rng:element>
        <rng:anyName/>
        <rng:ref name="__anyxml__"/>
      </rng:element>
      <rng:text/>
    </rng:choice>
  </rng:zeroOrMore>
</rng:define>
```

EXAMPLE: YANG statement

```
anyxml data {
    description "Any XML content allowed here.";
}
```

maps to the following fragment:

```
     <rng:element name="data">
         <a:documentation>Any XML content allowed here</a:documentation>
         <rng:ref name="__anyxml__"/>
     </rng:element>
```

10.2. The argument Statement

TOC

This statement is not mapped to the output schema, but see the rules for extension handling in Section 7.4 (YANG Language Extensions).

10.3. The augment Statement

TOC

As a substatement of 'uses', this statement is handled as a part of 'uses' mapping, see Section 10.54 (The uses Statement).
At the top level of a module or submodule, the 'augment' statement is used for augmenting the schema tree of another YANG module. If the latter module is not processed within the same mapping session, the top-level 'augment' statement MUST be ignored. Otherwise, the contents of the statement are added to the foreign module with the namespace of the module where the 'augment' statement appears.

10.4. The base Statement

TOC

This statement is ignored as a substatement of 'identity' and handled within the 'identityref' type if it appears as a substatement of that type definition, see Section 10.50.5 (The identityref Type).

10.5. The belongs-to Statement

TOC

This statement is not used since processing of submodules is always initiated from the main module, see Section 10.21 (The include Statement).

TOC

10.6. The bit Statement

This statement is handled within the "bits" type, see Section 10.50.3 (The bits Type).

10.7. The case Statement

TOC

This statement is mapped to <rng:group> element. If the argument of a sibling 'default' statement equals to ARGUMENT, @nma:default-case attribute with the value of "true" is added to that <rng:group> element.

10.8. The choice Statement

TOC

This statement is mapped to <rng:choice> element.
Unless 'choice' has the 'mandatory' substatement with the value of "true", the <rng:choice> element MUST be wrapped in <rng:optional>.

10.9. The config Statement

TOC

This statement is mapped to @nma:config attribute and ARGUMENT becomes its value.

10.10. The contact Statement

TOC

This statement is not used by the mapping since the output RELAX NG schema may result from multiple YANG modules created by different authors. The schema contains references to all input modules in the Dublin Core elements <dc:source>, see Section 10.31 (The module Statement). The original modules are the authoritative sources of the authorship information.

10.11. The container Statement

TOC

Using the procedure outlined in Section 7.1 (Optional and Mandatory Content), the mapping algorithm MUST determine whether the statement

defines an optional container, and if so, insert the <rng:optional> element and make it the new PARENT.
The container defined by this statement is then mapped to the <rng:element> element, which becomes a child of PARENT and uses ARGUMENT as the value of its @name attribute.

10.12. The default Statement

TOC

If this statement is a substatement of 'typedef' or 'leaf', it is mapped to the @nma:default attribute of PARENT and ARGUMENT becomes its value.
As a substatement of 'choice', the 'default' statement identifies the default case and is handled within the 'case' statement, see Section 10.7 (The case Statement). If the default case uses the shorthand notation where the 'case' statement is omitted, an extra <rng:group> element MUST be inserted with @nma:default-case attribute set to "true". The net result is then the same as if the 'case' statement wasn't omitted for the default case.
EXAMPLE. The following 'choice' statement

```
choice leaves {
    default feuille;
    leaf feuille { type empty; }
    leaf hoja { type empty; }
}
```

is mapped to

```
<rng:choice>
  <rng:group nma:default="true">
    <rng:element name="feuille">
      <rng:empty/>
    </rng:element>
  </rng:group>
  <rng:element name="hoja">
    <rng:empty/>
  </rng:element/>
</rng:choice>
```

10.13. The description Statement

TOC

This statement is ignored if it appears at the top level of each input YANG module. The description can be found in the source module that is

referred to by Dublin Core element <dc:source> and use ARGUMENT as its content.
Otherwise, this statement is mapped to the DTD compatibility element <a:documentation> and ARGUMENT becomes its text.
In order to get properly formatted in the RELAX NG compact syntax, this element SHOULD be inserted as the first child of PARENT.

10.14. The enum Statement

TOC

This statement is mapped to <rng:value> element and ARGUMENT becomes its text. All substatements except 'status' are ignored because the <rng:value> element cannot contain annotations, see [12] (ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. Second Edition.,” 12 2008.), Section 6.

10.15. The error-app-tag Statement

TOC

This statement is ignored unless it is a substatement of 'must'. In the latter case it is mapped to the <nma:error-app-tag> element. See also Section 10.32 (The must Statement).

10.16. The error-message Statement

TOC

This statement is ignored unless it is a substatement of 'must'. In the latter case it is mapped to the <nma:error-message> element. See also Section 10.32 (The must Statement).

10.17. The extension Statement

TOC

This statement is ignored. However, extensions to the YANG language MAY be mapped as described in Section 7.4 (YANG Language Extensions).

TOC

10.18. The grouping Statement

This statement is mapped to a RELAX NG named pattern definition <rng:define>, but only if the grouping defined by this statement is used *without refinements and augments* in at least one of the input modules. In this case, the named pattern definition becomes a child of the <rng:grammar> element and its name is ARGUMENT mangled according to the rules specified in Section 7.2 (Mapping YANG Groupings and Typedefs).
Whenever a grouping is used with additional refinements and/or augments, the grouping is expanded so that the refinements and augments may be applied directly to the prescribed schema nodes. See Section 7.2.1 (YANG Refinements and Augments) for further details and an example.
An implementation MAY offer the option of recording all 'grouping' statements as named patterns in the output RELAX NG schema even if they are not referenced. This is useful for mapping YANG "library" modules containing only 'typedef' and/or 'grouping' statements.

10.19. The identity Statement

TOC

This statement is not specifically mapped. However, if the identity defined by this statement is used as the base for an "identityref" type in any of the input modules, ARGUMENT will appear as the text of one of the <rng:value> elements in the mapping of that "identityref" type. See Section 10.50.5 (The identityref Type) for more details and an example.

10.20. The import Statement

TOC

This statement is not specifically mapped. The module whose name is in ARGUMENT has to be parsed so that the importing module be able to use its top-level groupings and typedefs and also augment the data tree of the imported module.
If the 'import' statement has the 'revision' substatement, the corresponding revision of the imported module MUST be used. The mechanism for finding a given module revision is outside the scope of this document.

TOC

10.21. The include Statement

This statement is not specifically mapped. The submodule whose name is in ARGUMENT has to be parsed and its contents mapped exactly as if the submodule text was a subset of the main module text.
If the 'include' statement has the 'revision' substatement, the corresponding revision of the submodule MUST be used. The mechanism for finding a given submodule revision is outside the scope of this document.

10.22. The input Statement

TOC

This statement is handled within 'rpc' statement, see Section 10.47 (The rpc Statement).

10.23. The key Statement

TOC

This statement is mapped to @nma:key attribute. ARGUMENT is MUST be translated so that every key is prefixed with the namespace prefix of the local module. The result of this translation then becomes the value of the @nma:key attribute.

10.24. The leaf Statement

TOC

This statement is mapped to the <rng:element> element and ARGUMENT becomes the value of its @name attribute.
The leaf is optional if there is no "mandatory true;" substatement and if the leaf is not declared among the keys of an enclosing list. In this case, the <rng:element> element MUST be wrapped in <rng:optional>.

10.25. The leaf-list Statement

TOC

This statement is mapped to a block enclosed by either <rng:zeroOrMore> or <rng:oneOrMore> element depending on whether the argument of 'min-elements' substatement is "0" or positive, respectively (it is zero by default). This <rng:zeroOrMore> or <rng:oneOrMore> element becomes the PARENT.
<rng:element> is the added as a child element of PARENT and ARGUMENT becomes the value of its @name attribute. If the 'leaf-list' statement

has the 'min-elements' substatement and its argument is greater than one, additional attribute @nma:min-elements is attached to <rng:element> and the argument of 'min-elements' becomes the value of this attribute. Similarly, if there is the 'max-elements' substatement and its argument value is not "unbounded", attribute @nma:max-elements is attached to this element and the argument of 'max-elements' becomes the value of this attribute.
EXAMPLE. YANG leaf-list

```
leaf-list foliage {
    min-elements 3;
    max-elements 6378;
    ordered-by user;
    type string;
}
```

is mapped to the following RELAX NG fragment:

```
<rng:oneOrMore>
  <rng:element name="foliage" nma:ordered-by="user"
               nma:min-elements="3" nma:max-elements="6378">
    <rng:data type="string"/>
  </rng:element>
</rng:oneOrMore>
```

10.26. The length Statement

TOC

This statement is handled within the "string" type, see Section 10.50.9 (The string Type).

10.27. The list Statement

TOC

This statement is mapped exactly as the 'leaf-list' statement, see Section 10.25 (The leaf-list Statement).

10.28. The mandatory Statement

TOC

This statement may appear as a substatement of 'leaf', 'choice' or 'anyxml' statement. If ARGUMENT is "true", the parent data node is mapped as mandatory, see Section 7.1 (Optional and Mandatory Content).

10.29. The max-elements Statement

TOC

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.25 (The leaf-list Statement).

10.30. The min-elements Statement

TOC

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.25 (The leaf-list Statement).

10.31. The module Statement

TOC

This statement is not specifically mapped except that a <dc:source> element SHOULD be created as a child of <rng:grammar> and contain ARGUMENT as a reference to the input YANG module. See also Section 10.46 (The revision Statement).
With respect to the conceptual tree schema, substatements of 'module' MUST be mapped so that

- top level data elements be defined as children of the <nmt:top> element;
- elements mapped from 'rpc' statements be defined as children of the <nmt:rpc-methods> element;
- elements mapped from 'notification' statements be defined as children of the <nmt:notifications> element.

10.32. The must Statement

TOC

This statement is mapped to the <nma:must> element. It has one mandatory attribute @assert (with no namespace), which contains ARGUMENT transformed into a valid XPath expression (see Section 7.3 (Translation of XPath Expressions)). The <nma:must> element may get other subelements resulting from mapping 'error-app-tag' and 'error-message' substatements. Other substatements of 'must', i.e., 'description' and 'reference', are ignored.
EXAMPLE. YANG statement

```
     must 'current() <= ../max-lease-time' {
         error-message
             "The default-lease-time must be less than max-lease-time";
     }
```

is mapped to

```
     <nma:must assert="current()&lt;=../dhcp:max-lease-time">
       <nma:error-message>
         The default-lease-time must be less than max-lease-time
       </nma:error-message>
     </nma:must>
```

10.33. The namespace Statement

TOC

This statement is mapped to @xmlns:xxx attribute of the <rng:grammar> element where "xxx" is the namespace prefix specified by the sibling 'prefix' statement. ARGUMENT becomes the value of this attribute.

10.34. The notification Statement

TOC

This statement is mapped to the following subtree in the RELAX NG schema ("yam" is the prefix of the local YANG module):

```
     <rng:element name="nmt:notification">
       <rng:element name="yam:ARGUMENT">
         ...
       </rng:element>
     </rng:element>
```

Substatements of 'notification' are mapped under <rng:element name="yam:ARGUMENT">.
The <rng:element name="nmt:rpc-notification"> element is a child of <rng:element name="nmt:notifications">.

10.35. The ordered-by Statement

TOC

This statement is mapped to @nma:ordered-by attribute and ARGUMENT becomes the value of this attribute. See Section 10.25 (The leaf-list Statement) for an example.

10.36. The organization Statement

TOC

This statement is not used by the mapping since the output RELAX NG schema may result from multiple YANG modules authored by different parties. The schema contains references to all input modules in the Dublin Core elements <dc:source>, see Section 10.31 (The module Statement). The original modules are the authoritative sources of the authorship information.

10.37. The output Statement

TOC

This statement is handled within 'rpc' statement, see Section 10.47 (The rpc Statement).

10.38. The path Statement

TOC

This statement is handled within "leafref" type, see Section 10.50.7 (The leafref Type).

10.39. The pattern Statement

TOC

This statement is handled within "string" type, see Section 10.50.9 (The string Type).

10.40. The position Statement

TOC

This statement is ignored.

10.41. The prefix Statement

TOC

This statement is handled within the sibling 'namespace' statement, see Section 10.33 (The namespace Statement), or within the parent 'import' statement, see Section 10.20 (The import Statement). As a substatement of 'belongs-to' (in submodules), the 'prefix' statement is ignored.

10.42. The presence Statement

TOC

This statement influences the mapping of 'container' (Section 10.11 (The container Statement)): it makes the parent container optional, regardless of its content. See also Section 7.1 (Optional and Mandatory Content).

10.43. The range Statement

TOC

This statement is handled within numeric types, see Section 10.50.8 (The numeric Types).

10.44. The reference Statement

TOC

This statement is ignored if it appears at the top level of a module or submodule.
Otherwise, this statement is mapped to <a:documentation> element and its text is set to ARGUMENT prefixed with "See: ".

10.45. The require-instance Statement

TOC

This statement is handled within the types "leafref" (Section 10.50.7 (The leafref Type)) and "instance-identifier" (Section 10.50.6 (The instance-identifier Type)).

10.46. The revision Statement

TOC

The mapping uses only the most recent instance of the 'revision' statement, i.e., one with the latest date in ARGUMENT, which specifies the current revision of the input YANG module [5] (Bjorklund, M., Ed., “YANG - A data modeling language for NETCONF,” March 2009.). This date SHOULD be recorded, together with the name of the YANG module, in the corresponding Dublin Core element <dc:source> (see Section 10.31 (The module Statement)), for example in this form:

```
<dc:source>YANG module 'foo', revision 2009-01-19</dc:source>
```

The 'description' substatement of 'revision' is not used.

10.47. The rpc Statement

TOC

This statement is mapped to the following subtree in the RELAX NG schema ("yam" is the prefix of the local YANG module):

```
<rng:element name="nmt:rpc-method">
  <rng:element name="nmt:input">
    <rng:element name="yam:ARGUMENT">
      <!-- mapped content of 'input' -->
    </rng:element>
  </rng:element>
  <rng:element name="nmt:output">
    <!-- mapped content of 'output' -->
  </rng:element>
</rng:element>
```

As indicated by the comments, contents of the 'input' substatement (if any) are mapped under <rng:element name="yam:ARGUMENT">. Similarly, contents of the 'output' substatement are mapped under <rng:element name="nmt:output">. If there is no 'output' substatement, the <rng:element name="nmt:output"> MUST NOT be present.
The <rng:element name="nmt:rpc-method"> element is a child of <rng:element name="nmt:rpc-methods">.

10.48. The status Statement

TOC

This statement is mapped to @nma:status attribute and ARGUMENT becomes its value.

10.49. The submodule Statement

TOC

This statement is not specifically mapped. Its substatements are mapped as if they appeared directly in the module the submodule belongs to.

10.50. The type Statement

TOC

Most YANG built-in types have an equivalent in the XSD datatype library [16] (Biron, P. and A. Malhotra, “XML Schema Part 2: Datatypes Second Edition,” October 2004.) as shown in Table 3 (Selected datatypes from the W3C XML Schema Type Library).

YANG type	XSD type	Meaning
int8	byte	8-bit integer value
int16	short	16-bit integer value
int32	int	32-bit integer value
int64	long	64-bit integer value
uint8	unsignedByte	8-bit unsigned integer value
uint16	unsignedShort	16-bit unsigned integer value
uint32	unsignedInt	32-bit unsigned integer value
uint64	unsignedLong	64-bit unsigned integer value
float32	float	32-bit IEEE floating-point value
float64	double	64-bit IEEE floating-point value
string	string	character string
boolean	boolean	"true" or "false"
binary	base64Binary	binary data in base64 encoding

Table 3: Selected datatypes from the W3C XML Schema Type Library

Details about the mapping of individual YANG built-in types are given in the following subsections.

10.50.1. The empty Type

TOC

This type is mapped to <rng:empty/>.

10.50.2. The boolean and binary Types

TOC

These two built-in types do not allow any restrictions and are mapped simply by inserting <rng:data> element whose @type attribute is set to

ARGUMENT mapped according to Table 3 (Selected datatypes from the W3C XML Schema Type Library).

10.50.3. The bits Type

TOC

This type is mapped to <rng:list> and for each 'bit' substatement the following XML fragment is inserted as a child of <rng:list>:

```
<rng:optional>
  <rng:value>bit_name</rng:value>
</rng:optional>
```

where bit_name is the name of the bit as found in the argument of the corresponding 'bit' statement.

10.50.4. The enumeration and union Types

TOC

These types are mapped to <rng:choice> element.

10.50.5. The identityref Type

TOC

This type is mapped to <rng:choice> element with one or more <rng:value> subelements. Each of the <rng:value> subelements MUST have the @type attribute and its value set to "QName". One <rng:value> subelement with argument of the 'base' substatement as its text MUST always be present. In addition, one <rng:value> substatement MUST be added for each identity declared locally or in an imported module that has the argument of the 'base' substatement as its base identity.
All namespace prefixes that are used for identities from imported modules MUST be appropriately defined.
EXAMPLE (taken from [5] (Bjorklund, M., Ed., “YANG - A data modeling language for NETCONF,” March 2009.), Section 7.6.13). Consider the following two YANG modules:

```
     module crypto-base {
         namespace "http://example.com/crypto-base";
         prefix "crypto";

         identity crypto-alg {
         description
             "Base identity from which all crypto algorithms
              are derived.";
         }
     }

     module des {
         namespace "http://example.com/des";
         prefix "des";

         import "crypto-base" {
             prefix "crypto";
         }

         identity des {
             base "crypto:crypto-alg";
             description "DES crypto algorithm";
         }

         identity des3 {
             base "crypto:crypto-alg";
             description "Triple DES crypto algorithm";
         }
     }
```

If these two modules are imported to another module, leaf definition

```
     leaf crypto {
         type identityref {
             base "crypto:crypto-alg";
         }
     }
```

is mapped to

```
     <rng:element name="crypto">
       <rng:choice>
         <rng:value type="QName">crypto:crypto-alg</value>
         <rng:value type="QName">des:des</value>
         <rng:value type="QName">des:des3</value>
      </rng:choice>
     </rng:element>
```

The "crypto" and "des" prefixes will by typically defined via attributes of the <rng:grammar> element.

10.50.6. The instance-identifier Type

TOC

This type is mapped to <rng:data> element with @type attribute set to "string". In addition, empty <nma:instance-identifier> element MUST be inserted as a child of PARENT.
The 'require-instance' substatement, if it exists, is mapped to the @require-instance attribute of <nma:instance-identifier>.

10.50.7. The leafref Type

TOC

This type is mapped to <rng:data> element with @type attribute set to the type of the leaf given in the argument of 'path' substatement. In addition, <nma:leafref> element MUST be inserted as a child of PARENT. The argument value of the 'path' substatement is set as the text of this element.
The 'require-instance' substatement, if it exists, is mapped to the @require-instance attribute of <nma:leafref>.

10.50.8. The numeric Types

TOC

YANG built-in numeric types are "int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64", "float32" and "float64". They are mapped to <rng:data> element with @type attribute set to ARGUMENT mapped according to Table 3 (Selected datatypes from the W3C XML Schema Type Library).
All numeric types support the 'range' restriction, which is handled in the following way:

- If the range expression consists of a single range part, insert the pair of RELAX NG facets

  ```
  <rng:param name="minInclusive">...</rng:param>
  ```

  and

  ```
  <rng:param name="maxInclusive">...</rng:param>
  ```

  Their contents are the lower and upper bound of the range part, respectively. If the range part consists of a single number, both "minInclusive" and "maxInclusive" facets use this value as their content. If the lower bound is "min", the "minInclusive" facet is omitted and if the upper bound is "max", the "maxInclusive" facet is omitted.

*If the range expression has multiple parts separated by "|", then repeat the <rng:data> element once for every range part and wrap them all in <rng:choice> element. Each <rng:data> element contains the "minInclusive" and "maxInclusive" facets for one part of the range expression as described in the previous item.

For example, the 'typedef' statement

```
typedef rt {
  type int32 {
    range "-6378..0|42|100..max";
  }
}
```

appearing at the top level of the "example" module is mapped to the following RELAX NG fragment:

```
<rng:define name="example__rt">
  <rng:choice>
    <rng:data type="int">
      <rng:param name="minInclusive">-6378</rng:param>
      <rng:param name="maxInclusive">0</rng:param>
    </rng:data>
    <rng:data type="int">
      <rng:param name="minInclusive">42</rng:param>
      <rng:param name="maxInclusive">42</rng:param>
    </rng:data>
    <rng:data type="int">
      <rng:param name="minInclusive">100</rng:param>
    </rng:data>
  </rng:choice>
</rng:define>
```

10.50.9. The string Type

This type is mapped to <rng:data> element with the @type attribute set to "string".
For the 'pattern' restriction, insert <rng:param> element with @name attribute set to "pattern". The argument of the 'pattern' statement (regular expression) becomes the content of this element.

The 'length' restriction is handled in the same way as the 'range' restriction for the numeric types, with the additional twist that if the length expression has multiple parts, the "pattern" facet

```
<rng:param name="pattern">...</rng:param>
```

if there is any, must be repeated inside each copy of the <rng:data> element, i.e., for each length part.

10.50.10. Derived Types

TOC

If the 'type' statement refers to a derived type, it is mapped in one of the following ways depending on whether it contains any restrictions as its substatements:

1. Without restrictions, the 'type' statement is mapped simply to the <rng:ref> element, i.e., a reference to a named pattern. If the RELAX NG definition of this named pattern has not been added to the output schema yet, the corresponding 'typedef' must be found and its mapping installed as a subelement of <rng:grammar>, see Section 10.51 (The typedef Statement). Even if a given derived type is used more than once in the input YANG modules, the mapping of the corresponding 'typedef' MUST be installed only once.

2. If any restrictions are present, the base type for the given derived type must be determined and the mapping of this base type is used. Restrictions appearing at all stages of the derivation chain must be taken into account and their conjunction added to the <rng:data> element which defines the basic type.

See Section 7.2.2 (Type derivation chains) for more details and an example.

10.51. The typedef Statement

TOC

This statement is mapped to a RELAX NG named pattern definition <rng:define>, but only if the type defined by this statement is used *without restrictions* in at least one of the input modules. In this case, the named pattern definition becomes a child of the <rng:grammar> element and its name is ARGUMENT mangled according to the rules specified in Section 7.2 (Mapping YANG Groupings and Typedefs).

Whenever a derived type is used with additional restrictions, the the base type for the derived type is used instead with restrictions (facets) that are a combination of all restrictions specified along the type derivation chain. See Section 10.50.10 (Derived Types) for further details and an example.
An implementation MAY offer the option of recording all 'typedef' statements as named patterns in the output RELAX NG schema even if they are not referenced. This is useful for mapping YANG "library" modules containing only 'typedef' and/or 'grouping' statements.

10.52. The unique Statement

TOC

This statement is mapped to @nma:unique attribute. ARGUMENT is translated so that every node identifier in each of its components is prefixed with the namespace prefix of the local module, unless the prefix is already present. The result of this translation then becomes the value of the @nma:unique attribute.
For example, assuming that the local module prefix is "ex",

```
unique "foo ex:bar/baz"
```

is mapped to the following attribute/value pair:

```
nma:unique="ex:foo ex:bar/ex:baz"
```

10.53. The units Statement

TOC

This statement is mapped to @nma:units attribute and ARGUMENT becomes its value.

10.54. The uses Statement

TOC

If this statement has neither 'refine' nor 'augment' substatements, it is mapped to <rng:ref> element and the value of its @name attribute is set to ARGUMENT mangled according to Section 7.2 (Mapping YANG Groupings and Typedefs)
If there are any 'refine' or 'augment' substatements, the corresponding grouping must be looked up and its contents is inserted as children of PARENT. See Section 7.2.1 (YANG Refinements and Augments) for further details and an example.

10.55. The value Statement

TOC

This statement is ignored.

10.56. The when Statement

TOC

This statement is mapped to @nma:when attribute and ARGUMENT becomes it value.

10.57. The yang-version Statement

TOC

This statement is not mapped to the output schema. However, an implementation SHOULD check that it is compatible with the YANG version declared by the statement (currently version 1).

10.58. The yin-element Statement

TOC

This statement is not mapped to the output schema, but see the rules for extension handling in Section 7.4 (YANG Language Extensions).

11. Mapping NETMOD-specific annotations to DSDL Schema Languages

TOC

This section contains mapping specification for individual NETMOD-specific annotations. In each case, the result of the mapping must be inserted into an appropriate context of the target DSDL schema as described in Section 8 (Mapping Conceptual Tree Schema to DSDL). The context is determined by the element definition in the conceptual tree schema to which the annotation is attached. In the rest of this section, we will denote CONTELEM the name of this context element properly qualified with its namespace prefix. Unless otherwise stated, Schematron asserts are descendants of the "standard" pattern and therefore active in both validation phases.

TOC

11.1. The @nma:config Annotation

This annotation MUST be observed when generating any schema for the reply to <nc:get-config>. In particular:

- *When generating RELAX NG, the contents of the CONTELEM definition MUST be changed to <rng:notAllowed>.

- *When generating Schematron or DSRL, the CONTELEM definition and all its descendants in the conceptual tree schema MUST be ignored.

11.2. The @nma:default Annotation

TBD

11.3. The @nma:default-case Annotation

TBD

11.4. The <nma:error-app-tag> Annotation

This annotation currently has no mapping defined.

11.5. The <nma:error-message> Annotation

This annotation is handled within <nma:must>, see Section 11.11 (The <nma:must> Annotation).

11.6. The <nma:instance-identifier> Annotation

This annotation currently has no mapping defined.
[Editor's note: The mapping is probably not possible with XPath 1.0 as the query language in Schematron. Shall we use EXSLT or XPath 2.0?]

11.7. The @nma:key Annotation

TOC

Assume this annotation has the value "k_1 k_2 ... k_n", i.e., specifies n child leaves as keys. The annotation is then mapped to the following Schematron report:

```
<sch:report test="CONDITION">
  Duplicate key of list "CONTELEM"
</sch:report>
```

where CONDITION has this form:

```
preceding-sibling::CONTELEM[C_1 and C_2 and ... and C_n]
```

Each C_i, for i=1,2,...,n, specifies the condition for violation of uniqueness of key k_i, namely

```
k_i=current()/k_i
```

11.8. The <nma:leafref> Annotation

TOC

The mapping of this annotation depends on its @require-instance attribute. If this attribute is not present or its value is "true", the referred leaf must exist in the instance document (this is verified by the RELAX NG schema) and the <nma:leafref> annotation is mapped to the following assert:

```
<sch:assert test="PATH=..">
  Leafref "CONTELEM" must have the same value as "PATH"
</sch:assert>
```

where PATH is the content of <nma:leafref>.
If the @require-instance attribute has the value "false", then the equality in contents of the context element and the referred leaf is required only if the referred leaf exists. Hence, <nma:leafref> is mapped to the following assert:

```
<sch:assert test="not(PATH) or PATH=..">
  Leafref "CONTELEM" must have the same value as "PATH"
</sch:assert>
```

In both cases the assert is a descendant of the "ref-integrity" pattern, which means that it will be used only for the "full" validation phase.

11.9. The @nma:min-elements Annotation

TOC

This annotation is mapped to the following Schematron assert:

```
     <sch:assert test="count(../CONTELEM)&gt;=MIN">
       List "CONTELEM" - item count must be at least MIN
     </sch:assert>
```

where MIN is the value of @nma:min-elements.

11.10. The @nma:max-elements Annotation

TOC

This annotation is mapped to the following Schematron assert:

```
     <sch:assert test="count(../CONTELEM)&lt;=MAX">
       List "CONTELEM" - item count must be at most MAX
     </sch:assert>
```

where MAX is the value of @nma:min-elements.

11.11. The <nma:must> Annotation

TOC

This annotation is mapped to the following Schematron assert:

```
     <sch:assert test="EXPRESSION">
       MESSAGE
     </sch:assert>
```

where EXPRESSION is the value of the mandatory @assert attribute of <nma:must>. If the <nma:error-message> subelement exists, MESSAGE is set to its content, otherwise it is set to the default message "Condition EXPRESSION must be true".

11.12. The <nma:ordered-by> Annotation

TOC

This annotation currently has no mapping defined.

TOC

11.13. The <nma:status> Annotation

This annotation currently has no mapping defined.

11.14. The @nma:unique Annotation

TOC

The mapping of this annotation is almost identical as for @nma:key, see Section 11.7 (The @nma:key Annotation), with two small differences:

- The value of @nma:unique is a list of descendant schema node identifiers rather than simple leaf names. However, the XPath expressions specified in Section 11.7 (The @nma:key Annotation) work without any modifications if the descendant schema node identifiers are substituted for k_1, k_2, ..., k_n.

- The message appearing as the text of <sch:report> is different: "Violated uniqueness for list CONTELEM".

11.15. The @nma:when Annotation

TOC

```
<sch:assert test="EXPRESSION or not(..)">
  Node "CONTELEM" requires "EXPRESSION"
</sch:assert>
```

where EXPRESSION is the value of @nma:when.

---

12. IANA Considerations

TOC

This document registers two namespace URIs in the IETF XML registry [22] (Mealling, M., “The IETF XML Registry,” January 2004.):

URI: urn:ietf:params:xml:ns:netmod:dsdl-annotations:1

URI: urn:ietf:params:xml:ns:netmod:conceptual-tree:1

---

13. References

TOC

[1]	Enns, R., "NETCONF Configuration Protocol," RFC 4741, December 2006 (TXT).
[2]	Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)," STD 15, RFC 1157, May 1990 (TXT).
[3]	McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)," STD 58, RFC 2578, April 1999 (TXT).
[4]	Elliott, C., Harrington, D., Jason, J., Schoenwaelder, J., Strauss, F., and W. Weiss, "SMIng Objectives," RFC 3216, December 2001 (TXT).
[5]	Bjorklund, M., Ed., "YANG - A data modeling language for NETCONF," draft-ietf-netmod-yang-04 (work in progress), March 2009 (HTML).
[6]	ISO/IEC, "Document Schema Definition Languages (DSDL) - Part 1: Overview," ISO/IEC 19757-1, 11 2004.
[7]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT).
[8]	Clark, J., Ed. and M. Murata, Ed., "RELAX NG DTD Compatibility," OASIS Committee Specification 3 December 2001, December 2001.
[9]	Kunze, J., "The Dublin Core Metadata Element Set," RFC 5013, August 2007 (TXT).
[10]	Chisholm, S. and H. Trevino, "NETCONF Event Notifications," RFC 5277, July 2008 (TXT).
[11]	ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 8: Document Semantics Renaming Language - DSRL," ISO/IEC 19757-8:2008(E), 12 2008.
[12]	ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. Second Edition.," ISO/IEC 19757-2:2008(E), 12 2008.
[13]	ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron," ISO/IEC 19757-3:2006(E), 6 2006.
[14]	Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition," World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004 (HTML).
[15]	Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition)," World Wide Web Consortium Recommendation REC-xml-20060816, August 2006 (HTML).
[16]	Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition," World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004 (HTML).

[17]	ISO/IEC, “Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. AMENDMENT 1: Compact Syntax,” ISO/IEC 19757-2:2003/ Amd. 1:2006(E), 1 2006.
[18]	Clark, J. and S. DeRose, “XML Path Language (XPath) Version 1.0,” World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999 (HTML).
[19]	Clark, J., “XSL Transformations (XSLT) Version 1.0,” World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999.
[20]	Schoenwaelder, J., Ed., “Common YANG Data Types,” draft-ietf-netmod-yang-types-01 (work in progress), November 2008 (HTML).
[21]	van der Vlist, E., “RELAX NG,” O'Reilly , 2004.
[22]	Mealling, M., “The IETF XML Registry,” BCP 81, RFC 3688, January 2004 (TXT).

Appendix A. RELAX NG Schema for NETMOD-specific Annotations

TOC

This appendix contains the RELAX NG schema for the NETMOD-specific annotations in both XML and compact syntax.
[Editor's note: It is currently only a set of named pattern definitions as templates for the annotation elements and attributes. We should find a way how to connect this to the schema for RELAX NG, which these annotations extend. One option may be NVDL or it can also be done as in the spec for DTD compatibility annotations.]

TOC

A.1. XML Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
ns="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<define name="config-attribute">
  <attribute name="config">
    <data type="boolean"/>
  </attribute>
</define>

<define name="default-attribute">
  <attribute name="default"/>
</define>

<define name="default-case-attribute">
  <attribute name="default-case">
    <data type="boolean"/>
  </attribute>
</define>

<define name="error-app-tag-element">
  <optional>
    <element name="error-app-tag">
      <text/>
    </element>
  </optional>
</define>

<define name="error-message-element">
  <optional>
    <element name="error-message">
      <text/>
    </element>
  </optional>
</define>

<define name="instance-identifier-element">
  <element name="instance-identifier">
    <optional>
      <attribute name="require-instance">
        <data type="boolean"/>
      </attribute>
    </optional>
  </element>
</define>

<define name="key-attribute">
  <attribute name="key">
```

```
    <list>
      <data type="QName"/>
    </list>
  </attribute>
</define>

<define name="leafref-element">
  <element name="leafref">
    <optional>
      <attribute name="require-instance">
        <data type="boolean"/>
      </attribute>
    </optional>
    <data type="string"/>
  </element>
</define>

<define name="min-elements-attribute">
  <attribute name="min-elements">
    <data type="integer"/>
  </attribute>
</define>

<define name="max-elements-attribute">
  <attribute name="max-elements">
    <data type="integer"/>
  </attribute>
</define>

<define name="must-element">
  <element name="must">
    <attribute name="assert">
      <data type="string"/>
    </attribute>
    <interleave>
      <ref name="err-app-tag-element"/>
      <ref name="err-message-element"/>
    </interleave>
  </element>
</define>

<define name="ordered-by-attribute">
  <attribute name="ordered-by">
    <choice>
      <value>user</value>
      <value>system</value>
    </choice>
  </attribute>
</define>
```

```
<define name="status-attribute">
  <attribute name="status">
    <choice>
      <value>current</value>
      <value>deprecated</value>
      <value>obsolete</value>
    </choice>
  </attribute>
</define>

<define name="unique-attribute">
  <attribute name="unique">
    <list>
      <data type="string"/>
    </list>
  </attribute>
</define>

<define name="units-attribute">
  <attribute name="units">
    <data type="string"/>
  </attribute>
</define>

<define name="when-attribute">
  <attribute name="when">
    <data type="string"/>
  </attribute>
</define>

</grammar>
```

TOC

A.2. Compact Syntax

```
     default namespace =
         "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"

     config-attribute = attribute config { xsd:boolean }
     default-attribute = attribute default { text }
     default-case-attribute = attribute default-case { xsd:boolean }
     error-app-tag-element = element error-app-tag { text }?
     error-message-element = element error-message { text }?
     instance-identifier-element =
       element instance-identifier {
         attribute require-instance { xsd:boolean }?
       }
     key-attribute =
       attribute key {
         list { xsd:QName }
       }
     leafref-element =
       element leafref {
         attribute require-instance { xsd:boolean }?,
         xsd:string
       }
     min-elements-attribute = attribute min-elements { xsd:integer }
     max-elements-attribute = attribute max-elements { xsd:integer }
     must-element =
       element must {
         attribute assert { xsd:string },
         (err-app-tag-element & err-message-element)
       }
     ordered-by-attribute = attribute ordered-by { "user" | "system" }
     status-attribute =
       attribute status { "current" | "deprecated" | "obsolete" }
     unique-attribute =
       attribute unique {
         list { xsd:string }
       }
     units-attribute = attribute units { xsd:string }
     when-attribute = attribute when { xsd:string }
```

Appendix B. Schema-Independent Library

TOC

In order to avoid copying the same named pattern definitions to the RELAX NG schemas generated in the second mapping step, we collected these definitions to a library file - schema-independent library -

which is included by the validating schemas under the file name "relaxng-lib.rng" (XML syntax) and "relaxng-lib.rnc" (compact syntax). The included definitions cover patterns for common elements from base NETCONF [1] (Enns, R., “NETCONF Configuration Protocol,” December 2006.) and event notifications [10] (Chisholm, S. and H. Trevino, “NETCONF Event Notifications,” July 2008.).

B.1. XML Syntax

TOC

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Library of RELAX NG pattern definitions -->

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
         xmlns:en="urn:ietf:params:xml:ns:netconf:notification:1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="message-id-attribute">
    <attribute name="message-id">
      <data type="string">
        <param name="maxLength">4095</param>
      </data>
    </attribute>
  </define>

  <define name="ok-element">
    <element name="nc:ok">
      <empty/>
    </element>
  </define>

  <define name="eventTime-element">
    <element name="en:eventTime">
      <data type="dateTime"/>
    </element>
  </define>
</grammar>
```

TOC

B.2. Compact Syntax

```
# Library of RELAX NG pattern definitions

namespace en = "urn:ietf:params:xml:ns:netconf:notification:1.0"
namespace nc = "urn:ietf:params:xml:ns:netconf:base:1.0"

message-id-attribute =
  attribute message-id {
    xsd:string { maxLength = "4095" }
  }
ok-element = element nc:ok { empty }
eventTime-element = element en:eventTime { xsd:dateTime }
```

Appendix C. Mapping DHCP Data Model - A Complete Example

TOC

This appendix demonstrates both steps of the YANG-to-DSDL mapping applied to the "canonical" DHCP tutorial data model. The input (single) YANG module is shown in Appendix C.1 (Input YANG Module) and the output schemas in the following two subsections.
The conceptual tree schema was obtained by the "rng" plugin of the pyang tool and the validating DSDL schemas by XSLT stylesheets that are also part of pyang distribution. RELAX NG schemas are shown in both XML and compact syntax. The latter was obtained from the former by using the Trang tool
Due to the limit of 72 characters per line, few long strings required manual editing, in particular the regular expression patterns for IP addresses etc. in the RELAX NG schemas. In the compact syntax we broke the patterns to appropriate segments and joined them with the concatenation operator "~". In the XML syntax, though, the long patterns had to be replaced by the placeholder string "... regex pattern ...". Also, line breaks were added to several documentation strings and Schematron messages. Other than that, the results of the automatic translations were not changed.

TOC

C.1. Input YANG Module

```
module dhcp {
  namespace "http://example.com/ns/dhcp";
  prefix dhcp;

  import yang-types { prefix yang; }
  import inet-types { prefix inet; }

  organization
    "yang-central.org";
  description
    "Partial data model for DHCP, based on the config of
     the ISC DHCP reference implementation.";

  container dhcp {
    description
      "configuration and operational parameters for a DHCP server.";

    leaf max-lease-time {
      type uint32;
      units seconds;
      default 7200;
    }

    leaf default-lease-time {
      type uint32;
      units seconds;
      must '. <= ../dhcp:max-lease-time' {
        error-message
          "The default-lease-time must be less than max-lease-time";
      }
      default 600;
    }

    uses subnet-list;

    container shared-networks {
      list shared-network {
        key name;

        leaf name {
          type string;
        }
        uses subnet-list;
      }
    }

    container status {
      config false;
      list leases {
```

```
      key address;

      leaf address {
        type inet:ip-address;
      }
      leaf starts {
        type yang:date-and-time;
      }
      leaf ends {
        type yang:date-and-time;
      }
      container hardware {
        leaf type {
          type enumeration {
            enum "ethernet";
            enum "token-ring";
            enum "fddi";
          }
        }
        leaf address {
          type yang:phys-address;
        }
      }
    }
  }
}

grouping subnet-list {
  description "A reusable list of subnets";
  list subnet {
    key net;
    leaf net {
      type inet:ip-prefix;
    }
    container range {
      presence "enables dynamic address assignment";
      leaf dynamic-bootp {
        type empty;
        description
          "Allows BOOTP clients to get addresses in this range";
      }
      leaf low {
        type inet:ip-address;
        mandatory true;
      }
      leaf high {
        type inet:ip-address;
        mandatory true;
      }
```

```
          }

          container dhcp-options {
            description "Options in the DHCP protocol";
            leaf-list router {
              type inet:host;
              ordered-by user;
              reference "RFC 2132, sec. 3.8";
            }
            leaf domain-name {
              type inet:domain-name;
              reference "RFC 2132, sec. 3.17";
            }
          }

          leaf max-lease-time {
            type uint32;
            units seconds;
            default 7200;
          }
        }
      }
    }
```

C.2. Conceptual Tree Schema

TOC

TOC

C.2.1. XML Syntax

```xml
<?xml version="1.0" encoding="UTF-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    xmlns:dc="http://purl.org/dc/terms"
    xmlns:dhcp="http://example.com/ns/dhcp"
    xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
    xmlns:nmt="urn:ietf:params:xml:ns:netmod:conceptual-tree:1"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <dc:creator>Pyang 0.9.3, RELAX NG plugin</dc:creator>
  <dc:source>YANG module 'dhcp'</dc:source>
  <start>
    <element name="nmt:netmod-tree">
      <element name="nmt:top">
        <interleave>
          <optional>
            <element name="dhcp:dhcp">
              <a:documentation>
      configuration and operational parameters for a DHCP server.
              </a:documentation>
              <optional>
                <element name="dhcp:max-lease-time"
                         nma:default="7200" nma:units="seconds">
                  <data type="unsignedInt"/>
                </element>
              </optional>
              <optional>
                <element name="dhcp:default-lease-time"
                         nma:default="600" nma:units="seconds">
                  <data type="unsignedInt"/>
                  <nma:must
                      assert=". &lt;= ../dhcp:max-lease-time">
                    <nma:error-message>
        The default-lease-time must be less than max-lease-time
                    </nma:error-message>
                  </nma:must>
                </element>
              </optional>
              <ref name="_dhcp__subnet-list"/>
              <optional>
                <element name="dhcp:shared-networks">
                  <zeroOrMore>
                    <element name="dhcp:shared-network"
                             nma:key="dhcp:name">
                      <element name="dhcp:name">
                        <data type="string"/>
                      </element>
                      <ref name="_dhcp__subnet-list"/>
              </element>
            </zeroOrMore>
          </element>
        </optional>
        <optional>
          <element name="dhcp:status" nma:config="false">
            <zeroOrMore>
              <element name="dhcp:leases"
                       nma:key="dhcp:address">
                <element name="dhcp:address">
                  <ref name="inet-types__ip-address"/>
                </element>
                <optional>
                  <element name="dhcp:starts">
                    <ref name="yang-types__date-and-time"/>
                  </element>
                </optional>
                <optional>
                  <element name="dhcp:ends">
                    <ref name="yang-types__date-and-time"/>
                  </element>
                </optional>
                <optional>
                  <element name="dhcp:hardware">
                    <optional>
                      <element name="dhcp:type">
                        <choice>
                          <value>ethernet</value>
                          <value>token-ring</value>
                          <value>fddi</value>
                        </choice>
                      </element>
                    </optional>
                    <optional>
                      <element name="dhcp:address">
                        <ref name="yang-types__phys-address"/>
                      </element>
                    </optional>
                  </element>
                </optional>
              </element>
            </zeroOrMore>
          </element>
        </optional>
      </element>
    </optional>
  </interleave>
</element>
<element name="nmt:rpc-methods">
```

```
      <empty/>
    </element>
    <element name="nmt:notifications">
      <empty/>
    </element>
  </element>
</start>
<define name="_dhcp__subnet-list">
  <a:documentation>A reusable list of subnets</a:documentation>
  <zeroOrMore>
    <element name="dhcp:subnet" nma:key="dhcp:net">
      <element name="dhcp:net">
        <ref name="inet-types__ip-prefix"/>
      </element>
      <optional>
        <element name="dhcp:range">
          <optional>
            <element name="dhcp:dynamic-bootp">
              <a:documentation>
      Allows BOOTP clients to get addresses in this range
              </a:documentation>
              <empty/>
            </element>
          </optional>
          <element name="dhcp:low">
            <ref name="inet-types__ip-address"/>
          </element>
          <element name="dhcp:high">
            <ref name="inet-types__ip-address"/>
          </element>
        </element>
      </optional>
      <optional>
        <element name="dhcp:dhcp-options">
          <a:documentation>
            Options in the DHCP protocol
          </a:documentation>
          <zeroOrMore>
            <element name="dhcp:router" nma:ordered-by="user">
              <a:documentation>
                See: RFC 2132, sec. 3.8
              </a:documentation>
              <ref name="inet-types__host"/>
            </element>
          </zeroOrMore>
          <optional>
            <element name="dhcp:domain-name">
              <a:documentation>
                See: RFC 2132, sec. 3.17
              </a:documentation>
              <ref name="inet-types__domain-name"/>
            </element>
          </optional>
        </element>
      </optional>
      <optional>
        <element name="dhcp:max-lease-time"
                 nma:default="7200" nma:units="seconds">
          <data type="unsignedInt"/>
        </element>
      </optional>
    </element>
  </zeroOrMore>
</define>
<define name="inet-types__ip-prefix">
  <choice>
    <ref name="inet-types__ipv4-prefix"/>
    <ref name="inet-types__ipv6-prefix"/>
  </choice>
</define>
<define name="inet-types__ipv4-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="inet-types__ipv6-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="inet-types__ip-address">
  <choice>
    <ref name="inet-types__ipv4-address"/>
    <ref name="inet-types__ipv6-address"/>
  </choice>
</define>
<define name="inet-types__ipv4-address">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="inet-types__ipv6-address">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="inet-types__host">
  <choice>
```

```
      <ref name="inet-types__ip-address"/>
      <ref name="inet-types__domain-name"/>
    </choice>
  </define>
  <define name="inet-types__domain-name">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="yang-types__date-and-time">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="yang-types__phys-address">
    <data type="string"/>
  </define>
</grammar>
```

TOC

C.2.2. Compact Syntax

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace dc = "http://purl.org/dc/terms"
namespace dhcp = "http://example.com/ns/dhcp"
namespace nma = "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
namespace nmt = "urn:ietf:params:xml:ns:netmod:conceptual-tree:1"

dc:creator [ "Pyang 0.9.3, RELAX NG plugin" ]
dc:source [ "YANG module 'dhcp'" ]
start =
  element nmt:netmod-tree {
    element nmt:top {

      ## configuration and operational parameters for a DHCP server.
      element dhcp:dhcp {
        [ nma:default = "7200" nma:units = "seconds" ]
        element dhcp:max-lease-time { xsd:unsignedInt }?,
        [ nma:default = "600" nma:units = "seconds" ]
        element dhcp:default-lease-time {
          xsd:unsignedInt
          >> nma:must [
               assert = ". <= ../dhcp:max-lease-time"
               nma:error-message [
          "The default-lease-time must be less than max-lease-time"
               ]
             ]
        }?,
        _dhcp__subnet-list,
        element dhcp:shared-networks {
          [ nma:key = "dhcp:name" ]
          element dhcp:shared-network {
            element dhcp:name { xsd:string },
            _dhcp__subnet-list
          }*
        }?,
        [ nma:config = "false" ]
        element dhcp:status {
          [ nma:key = "dhcp:address" ]
          element dhcp:leases {
            element dhcp:address { inet-types__ip-address },
            element dhcp:starts { yang-types__date-and-time }?,
            element dhcp:ends { yang-types__date-and-time }?,
            element dhcp:hardware {
              element dhcp:type { "ethernet"
                                | "token-ring"
                                | "fddi"
                                }?,
              element dhcp:address { yang-types__phys-address }?
            }?
          }*
        }?
      }?
    },
    element nmt:rpc-methods { empty },
    element nmt:notifications { empty }
  }

## A reusable list of subnets
_dhcp__subnet-list =
  [ nma:key = "dhcp:net" ]
  element dhcp:subnet {
    element dhcp:net { inet-types__ip-prefix },
    element dhcp:range {

      ## Allows BOOTP clients to get addresses in this range
      element dhcp:dynamic-bootp { empty }?,
      element dhcp:low { inet-types__ip-address },
      element dhcp:high { inet-types__ip-address }
    }?,

    ## Options in the DHCP protocol
    element dhcp:dhcp-options {

      ## See: RFC 2132, sec. 3.8
      [ nma:ordered-by = "user" ]
      element dhcp:router { inet-types__host }*,

      ## See: RFC 2132, sec. 3.17
      element dhcp:domain-name { inet-types__domain-name }?
    }?,
    [ nma:default = "7200" nma:units = "seconds" ]
    element dhcp:max-lease-time { xsd:unsignedInt }?
  }*
inet-types__ip-prefix =
  inet-types__ipv4-prefix | inet-types__ipv6-prefix
inet-types__ipv4-prefix =
  xsd:string {
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)" ~
      "{3}([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])/\p{N}+"
  }
inet-types__ipv6-prefix =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})/" ~
      "\p{N}+)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\." ~
      "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))/\p{N}+)|" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*/\p{N}+)" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(([0-9]" ~
      "{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))/\p{N}+)"
  }
inet-types__ip-address =
  inet-types__ipv4-address | inet-types__ipv6-address
inet-types__ipv4-address =
  xsd:string {
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-1]?" ~
      "[0-9]?[0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}\p{L}]+)?"
  }
inet-types__ipv6-address =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})(%[\p{N}" ~
      "\p{L}]+)?)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\." ~
      "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]+)?)|" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(%[\p{N}" ~
      "\p{L}]+)?)((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*" ~
      "(::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(([0-9]{1,3}" ~
      "\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]+)?)"
  }
inet-types__host = inet-types__ip-address | inet-types__domain-name
inet-types__domain-name =
  xsd:string {
    pattern =
      "([a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]\.)*" ~
      "[a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]"
    pattern =
      "([r-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]\.)*" ~
      "[a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]"
  }
yang-types__date-and-time =
  xsd:string {
    pattern =
      "\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}" ~
      "(\.d*)?(Z|(\+|-)\d{2}:\d{2})"
  }
yang-types__phys-address = xsd:string
```

C.3. Final DSDL Schemas

This appendix contains DSDL schemas that were obtained from the conceptual tree schema in Appendix C.2 (Conceptual Tree Schema) by XSL transformations. These schemas can be directly used for validating a reply to unfiltered <get> with the contents corresponding to the DHCP data model.
The RELAX NG schema (again shown in both XML and compact syntax) includes the schema independent library from Appendix B (Schema-Independent Library).

TOC

C.3.1. RELAX NG Schema for <get> Reply - XML Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    xmlns:dc="http://purl.org/dc/terms"
    xmlns:dhcp="http://example.com/ns/dhcp"
    xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
    xmlns:nmt="urn:ietf:params:xml:ns:netmod:conceptual-tree:1"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
    ns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rng:include xmlns:rng="http://relaxng.org/ns/structure/1.0"
               href="./relaxng-lib.rng"/>
  <start>
    <rng:element xmlns:rng="http://relaxng.org/ns/structure/1.0"
                 name="rpc-reply">
      <rng:ref name="message-id-attribute"/>
      <rng:element name="data">
        <interleave>
          <optional>
            <element name="dhcp:dhcp">
              <optional>
                <element name="dhcp:max-lease-time">
                  <data type="unsignedInt"/>
                </element>
              </optional>
              <optional>
                <element name="dhcp:default-lease-time">
                  <data type="unsignedInt"/>
                </element>
              </optional>
              <ref name="_dhcp__subnet-list"/>
              <optional>
                <element name="dhcp:shared-networks">
                  <zeroOrMore>
                    <element name="dhcp:shared-network">
                      <element name="dhcp:name">
                        <data type="string"/>
                      </element>
                      <ref name="_dhcp__subnet-list"/>
                    </element>
                  </zeroOrMore>
                </element>
              </optional>
              <optional>
                <element name="dhcp:status">
                  <zeroOrMore>
                    <element name="dhcp:leases">
                      <element name="dhcp:address">
                      <ref name="inet-types__ip-address"/>
                    </element>
                    <optional>
                      <element name="dhcp:starts">
                        <ref name="yang-types__date-and-time"/>
                      </element>
                    </optional>
                    <optional>
                      <element name="dhcp:ends">
                        <ref name="yang-types__date-and-time"/>
                      </element>
                    </optional>
                    <optional>
                      <element name="dhcp:hardware">
                        <optional>
                          <element name="dhcp:type">
                            <choice>
                              <value>ethernet</value>
                              <value>token-ring</value>
                              <value>fddi</value>
                            </choice>
                          </element>
                        </optional>
                        <optional>
                          <element name="dhcp:address">
                            <ref name="yang-types__phys-address"/>
                          </element>
                        </optional>
                      </element>
                    </optional>
                  </element>
                </zeroOrMore>
              </element>
            </optional>
          </element>
        </optional>
      </interleave>
    </rng:element>
  </rng:element>
</start>
<define name="_dhcp__subnet-list">
  <zeroOrMore>
    <element name="dhcp:subnet">
      <element name="dhcp:net">
        <ref name="inet-types__ip-prefix"/>
      </element>
      <optional>
        <element name="dhcp:range">
          <optional>
```

```
            <element name="dhcp:dynamic-bootp">
              <empty/>
            </element>
          </optional>
          <element name="dhcp:low">
            <ref name="inet-types__ip-address"/>
          </element>
          <element name="dhcp:high">
            <ref name="inet-types__ip-address"/>
          </element>
        </element>
      </optional>
      <optional>
        <element name="dhcp:dhcp-options">
          <zeroOrMore>
            <element name="dhcp:router">
              <ref name="inet-types__host"/>
            </element>
          </zeroOrMore>
          <optional>
            <element name="dhcp:domain-name">
              <ref name="inet-types__domain-name"/>
            </element>
          </optional>
        </element>
      </optional>
      <optional>
        <element name="dhcp:max-lease-time">
          <data type="unsignedInt"/>
        </element>
      </optional>
    </element>
  </zeroOrMore>
</define>
<define name="inet-types__ip-prefix">
  <choice>
    <ref name="inet-types__ipv4-prefix"/>
    <ref name="inet-types__ipv6-prefix"/>
  </choice>
</define>
<define name="inet-types__ipv4-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="inet-types__ipv6-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
```

```
    </define>
    <define name="inet-types__ip-address">
      <choice>
        <ref name="inet-types__ipv4-address"/>
        <ref name="inet-types__ipv6-address"/>
      </choice>
    </define>
    <define name="inet-types__ipv4-address">
      <data type="string">
        <param name="pattern">... regex pattern ...</param>
      </data>
    </define>
    <define name="inet-types__ipv6-address">
      <data type="string">
        <param name="pattern">... regex pattern ...</param>
      </data>
    </define>
    <define name="inet-types__host">
      <choice>
        <ref name="inet-types__ip-address"/>
        <ref name="inet-types__domain-name"/>
      </choice>
    </define>
    <define name="inet-types__domain-name">
      <data type="string">
        <param name="pattern">... regex pattern ...</param>
        <param name="pattern">... regex pattern ...</param>
      </data>
    </define>
    <define name="yang-types__date-and-time">
      <data type="string">
        <param name="pattern">... regex pattern ...</param>
      </data>
    </define>
    <define name="yang-types__phys-address">
      <data type="string"/>
    </define>
  </grammar>
```

TOC

C.3.2. RELAX NG Schema for <get> Reply - Compact Syntax

```
default namespace = "urn:ietf:params:xml:ns:netconf:base:1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace dc = "http://purl.org/dc/terms"
namespace dhcp = "http://example.com/ns/dhcp"
namespace nma = "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
namespace nmt = "urn:ietf:params:xml:ns:netmod:conceptual-tree:1"

include "relaxng-lib.rnc"
start =
  element rpc-reply {
    message-id-attribute,
    element data {
      element dhcp:dhcp {
        element dhcp:max-lease-time { xsd:unsignedInt }?,
        element dhcp:default-lease-time { xsd:unsignedInt }?,
        _dhcp__subnet-list,
        element dhcp:shared-networks {
          element dhcp:shared-network {
            element dhcp:name { xsd:string },
            _dhcp__subnet-list
          }*
        }?,
        element dhcp:status {
          element dhcp:leases {
            element dhcp:address { inet-types__ip-address },
            element dhcp:starts { yang-types__date-and-time }?,
            element dhcp:ends { yang-types__date-and-time }?,
            element dhcp:hardware {
              element dhcp:type { "ethernet"
                                | "token-ring"
                                | "fddi"
                                }?,
              element dhcp:address { yang-types__phys-address }?
            }?
          }*
        }?
      }?
    }
  }
_dhcp__subnet-list =
  element dhcp:subnet {
    element dhcp:net { inet-types__ip-prefix },
    element dhcp:range {
      element dhcp:dynamic-bootp { empty }?,
      element dhcp:low { inet-types__ip-address },
      element dhcp:high { inet-types__ip-address }
    }?,
    element dhcp:dhcp-options {
      element dhcp:router { inet-types__host }*,
      element dhcp:domain-name { inet-types__domain-name }?
    }?,
    element dhcp:max-lease-time { xsd:unsignedInt }?
  }*
inet-types__ip-prefix =
  inet-types__ipv4-prefix | inet-types__ipv6-prefix
inet-types__ipv4-prefix =
  xsd:string {
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.)" ~
      "{3}([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])/\p{N}+"
  }
inet-types__ipv6-prefix =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})/" ~
      "\p{N}+)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\." ~
      "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))/\p{N}+)|" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*/\p{N}+)" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(([0-9]" ~
      "{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))/\p{N}+)"
  }
inet-types__ip-address =
  inet-types__ipv4-address | inet-types__ipv6-address
inet-types__ipv4-address =
  xsd:string {
    pattern =
      "(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-1]?" ~
      "[0-9]?[0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}\p{L}]+)?"
  }
inet-types__ipv6-address =
  xsd:string {
    pattern =
      "((([0-9a-fA-F]{1,4}:){7})([0-9a-fA-F]{1,4})(%[\p{N}" ~
      "\p{L}]+)?)|((([0-9a-fA-F]{1,4}:){6})(([0-9]{1,3}\." ~
      "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]+)?)|" ~
      "((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(::)" ~
      "(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(%[\p{N}" ~
      "\p{L}]+)?)((([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*" ~
      "(::)(([0-9a-fA-F]{1,4}:)*([0-9a-fA-F]{1,4}))*(([0-9]{1,3}" ~
      "\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(%[\p{N}\p{L}]+)?)"
  }
inet-types__host = inet-types__ip-address | inet-types__domain-name
inet-types__domain-name =
  xsd:string {
    pattern =
          "([a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]\.)*" ~
          "[a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]"
        pattern =
          "([r-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]\.)*" ~
          "[a-zA-Z][a-zA-Z0-9\-]*[a-zA-Z0-9]"
      }
    yang-types__date-and-time =
      xsd:string {
        pattern =
          "\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}" ~
          "(\.d*)?(Z|(\+|-)\d{2}:\d{2})"
      }
    yang-types__phys-address = xsd:string
```

C.4. Schematron Schema for <get> Reply

```
<?xml version="1.0" encoding="utf-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:ns uri="http://example.com/ns/dhcp" prefix="dhcp"/>
  <sch:ns uri="urn:ietf:params:xml:ns:netconf:base:1.0"
          prefix="nc"/>
  <sch:phase id="full">
    <sch:active pattern="standard"/>
    <sch:active pattern="ref-integrity"/>
  </sch:phase>
  <sch:phase id="noref">
    <sch:active pattern="standard"/>
  </sch:phase>
  <sch:pattern id="standard">
    <sch:rule id="std-id2246197" abstract="true">
      <sch:report test="preceding-sibling::dhcp:subnet
                        [dhcp:net=current()/dhcp:net]">
        Duplicate key of list dhcp:subnet
      </sch:report>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:default-lease-time">
      <sch:assert test=". &lt;= ../dhcp:max-lease-time">
        The default-lease-time must be less than max-lease-time
      </sch:assert>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:subnet">
      <sch:extends rule="std-id2246197"/>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:shared-networks/dhcp:shared-network">
      <sch:report test="preceding-sibling::dhcp:shared-network
                        [dhcp:name=current()/dhcp:name]">
        Duplicate key of list dhcp:shared-network
      </sch:report>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:shared-networks/dhcp:shared-network/
                       dhcp:subnet">
      <sch:extends rule="std-id2246197"/>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:status/dhcp:leases">
      <sch:report test="preceding-sibling::dhcp:leases
                        [dhcp:address=current()/dhcp:address]">
        Duplicate key of list dhcp:leases
      </sch:report>
    </sch:rule>
  </sch:pattern>
```

```
      <sch:pattern id="ref-integrity"/>
    </sch:schema>
```

C.5. DSRL Schema for <get> Reply

TOC

TBD

Appendix D. Change Log

TOC

D.1. Changes Between Versions -00 and -01

TOC

- *Attributes @nma:min-elements and @nma:max-elements are attached to <rng:element> (list entry) and not to <rng:zeroOrMore> or <rng:oneOrMore>.

- *Keys and all node identifiers in 'key' and 'unique' statements are prefixed.

- *Fixed the mapping of 'rpc' and 'notification'.

- *Removed previous Sec. 7.5 "RPC Signatures and Notifications" - the same information is now contained in Section 10.47 (The rpc Statement) and Section 10.34 (The notification Statement).

- *Added initial "_" to mangled names of groupings.

- *Mandated the use of @xmlns:xxx as the only method for declaring the target namespace.

- *Added section "Handling of XML Namespaces" to explain the previous item.

- *Completed DHCP example in Appendix C (Mapping DHCP Data Model - A Complete Example).

- *Almost all text about the second mapping step is new.

Authors' Addresses

TOC

	Ladislav Lhotka
	CESNET
Email:	lhotka@cesnet.cz
	Rohan Mahy
	Plantronics
Email:	rohan@ekabal.com
	Sharon Chisholm
	Nortel
Email:	schishol@nortel.com