

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 13, 2010

A. Bierman
InterWorking Labs
June 11, 2010

NETCONF System Monitoring
draft-bierman-netconf-system-monitoring-00

Abstract

The NETCONF protocol provides mechanisms to manipulate configuration datastores. However, client applications often need to examine system information to determine the appropriate configuration requirements. In addition, common system events such as a change in system capabilities may impact management applications. Standard mechanisms are needed to support the monitoring of the managed system supported by a NETCONF server. This document defines a YANG module for the monitoring of system information, which allows a NETCONF client to identify system properties and receive notifications for system events.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 13, 2010.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 1.1. Terminology | 3 |
| 2. YANG Module for System Monitoring | 3 |
| 2.1. Overview | 3 |
| 2.1.1. <system> Container | 4 |
| 2.1.2. Notifications | 4 |
| 2.2. Definitions | 4 |
| 3. IANA Considerations | 14 |
| 4. Security Considerations | 15 |
| 5. Acknowledgements | 15 |
| 6. References | 15 |
| 6.1. Normative References | 15 |
| 6.2. Informative References | 16 |
| Appendix A. Change Log | 16 |
| A.1. 00 | 16 |
| Author's Address | 16 |

1. Introduction

The NETCONF protocol [RFC4741] provides mechanisms to manipulate configuration datastores. However, client applications often need to examine system information to determine the appropriate configuration requirements. In addition, common system events such as a change in system capabilities may impact management applications. Standard mechanisms are needed to support the monitoring of the managed system supported by a NETCONF server. This document defines a YANG module [I-D.ietf-netmod-yang] for the monitoring of system information, which allows a NETCONF client to identify system properties and receive notifications [RFC5277] for system events.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [RFC4741]:

- o client
- o datastore
- o operation
- o server

The following terms are defined in [RFC5277]:

- o event
- o stream
- o subscription

The following term is defined in [I-D.ietf-netmod-yang]:

- o data node

2. YANG Module for System Monitoring

2.1. Overview

The following YANG module defines data node definitions suitable for use with NETCONF operations such as <get>, <get-config>, and <copy-config>. In addition, a small number of system events are defined for use within the NETCONF stream, and accessible to clients via the subscription mechanism in [RFC5277].

The YANG language is defined in [I-D.ietf-netmod-yang]. The NETCONF operations are defined in YANG in [RFC4741].

2.1.1.1. <system> Container

The <system> element provides commonly used vendor-specific information to identify and control a managed system:

- o sys-name: The name of the managed system.
- o sys-current-date-and-time: The current time as known to the managed system.
- o sys-boot-date-and-time: The time when the system last restarted.
- o sys-server-id: A vendor-specific string identifying the NETCONF server implementation.
- o uname: A container of common system naming information, such as the release, version, machine, and nodename of the system.

2.1.1.2. Notifications

This module defines some system events to notify a client application that the system state has changed.

- o sys-startup: Generated during a system restart. Lists any errors that were encountered while loading the <running> datastore during system initialization.
- o sys-config-change: Generated when the <running> configuration datastore is changed. Summarizes each edit being reported.
- o sys-capability-change: Generated when the NETCONF server capabilities are changed. Indicates which capabilities have been added, deleted, and/or modified.
- o sys-session-start: Generated when the NETCONF session is started. Indicates the identity of the user that started the session.
- o sys-session-end: Generated when the NETCONF session is terminated. Indicates the identity of the user that owned the session, and why the session was terminated.
- o sys-conformed-commit: Generated when the NETCONF confirmed-commit event occurs. Indicates the current state of the confirmed-commit operation in progress.

2.2. Definitions

```
<CODE BEGINS> file="netconf-system@2010-06-10.yang"
```

```
module netconf-system {  
  
    namespace "urn:ietf:params:xml:ns:yang:netconf-system";  
  
    prefix ncsys;  
  
    import ietf-yang-types { prefix yang; }  
    import ietf-inet-types { prefix inet; }
```

organization

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

contact

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

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

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

Editor: Andy Bierman
<<mailto:andyb@iwl.com>>;

description

"This module defines an YANG data model for use with the NETCONF protocol that allows the NETCONF client to monitor common system information and receive common system events.

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

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

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

// RFC Ed.: replace XXXX with actual RFC number and remove this note

// RFC Ed.: remove this note

// Note: extracted from draft-bierman-netconf-system-00.txt

revision 2010-06-10 {

description

"Initial version.";

reference

"RFC XXXX: NETCONF System Monitoring";

}

// RFC Ed.: replace XXXX with actual

// RFC number and remove this note

typedef error-type-type {

```
description "NETCONF Error Type";
type enumeration {
  enum transport {
    description "Transport layer error";
  }
  enum rpc {
    description "Operation layer error";
  }
  enum protocol {
    description "Protocol layer error";
  }
  enum application {
    description "Application layer error";
  }
}

typedef error-tag-type {
  description "NETCONF Error Tag";
  type enumeration {
    // descriptions TBD; normative text in RFC 4741
    enum in-use;
    enum invalid-value;
    enum too-big;
    enum missing-attribute;
    enum bad-attribute;
    enum unknown-attribute;
    enum missing-element;
    enum bad-element;
    enum unknown-element;
    enum unknown-namespace;
    enum access-denied;
    enum lock-denied;
    enum resource-denied;
    enum rollback-failed;
    enum data-exists;
    enum data-missing;
    enum operation-not-supported;
    enum operation-failed;
    enum partial-operation;
  }
}

typedef error-severity-type {
  description "NETCONF Error Severity";
  type enumeration {
    enum error {
      description "Error severity";
    }
  }
}
```

```
    }
    enum warning {
        description "Warning severity";
    }
}

typedef edit-operation-type {
    description
        "NETCONF 'operation' Attribute values";
    type enumeration {
        enum merge;
        enum replace;
        enum create;
        enum delete;
    }
    default "merge";
}

grouping sys-common-session-parms {

    leaf user-name {
        description
            "Name of the user for the session.";
        type string;
    }

    leaf session-id {
        description "Identifier of the session.";
        type uint32;    // nc:session-id-or-zero-type;
        mandatory true;
    }

    leaf remote-host {
        description
            "Address of the remote host for the session.";
        type inet:ip-address;
    }
}

container system {
    description
        "Basic objects for NETCONF system identification.";

    config false;

    leaf sys-name {
        description "The system name.";
    }
}
```

```
    reference "RFC 3418, sysName object";
    type string;
    mandatory true;
}

leaf sys-current-date-time {
    description
        "The current system date and time.";
    type yang:date-and-time;
    mandatory true;
}

leaf sys-boot-date-time {
    description
        "The system date and time when the system
        last restarted.";
    type yang:date-and-time;
    mandatory true;
}

leaf sys-server-id {
    description
        "The vendor-specific name and version ID string
        for the NETCONF server running on this system.";
    type string;
    mandatory true;
}

container uname {
    description
        "Contains the broken out fields from the
        output of the 'uname' command on this machine.";
    leaf sysname {
        type string;
        description
            "The name of the operating system in use.";
    }

    leaf release {
        type string;
        description
            "The current release level of the operating
            system in use.";
    }

    leaf version {
        type string;
        description

```



```
        "The current version level of the operating
        system in use.";
    }

    leaf machine {
        type string;
        description "A description of the hardware in use.";
    }

    leaf nodename {
        type string;
        description
            "The host name of this system, as reported by
            the uname command.";
    }
} // container uname
} // container system

notification sys-startup {
    description
        "Generated when the system restarts.
        Used for logging purposes, since no
        sessions are actually active when
        the system restarts.";

    leaf startup-source {
        description
            "The system-specific filespec used to load the
            running configuration. This leaf will only be
            present if there was a startup configuration file used.";
        type string;
    }
}

list boot-error {
    description
        "There will be one entry for each <rpc-error>
        encountered during the load config operation.
        There is no particular order, so no key is defined.
        This list will only be present if the server is configured
        to continue on error during startup, and there were recoverable
        errors encountered during the last restart of the server.";

    leaf error-type {
        description
            "Defines the conceptual layer that the error occurred.";
        type error-type-type;
        mandatory true;
    }
}
```

```
    }

    leaf error-tag {
      description
        "Contains a string identifying the error condition.";
      type error-tag-type;
      mandatory true;
    }

    leaf error-severity {
      description
        "Contains a string identifying the error severity, as
        determined by the device.";
      type error-severity-type;
      mandatory true;
    }

    leaf error-app-tag {
      description
        "Contains a string identifying the data-model-specific
        or implementation-specific error condition, if one exists.";
      type string;
    }

    leaf error-path {
      description
        "Contains the absolute XPath expression identifying
        the element path to the node that is associated with
        the error being reported in a particular <rpc-error>
        element.";
      type yang:xpath1.0;
    }

    leaf error-message {
      description
        "Contains a string suitable for human display that
        describes the error condition.";
      type string;    // LangString;
    }

    anyxml error-info {
      description
        "Contains protocol- or data-model-specific error content.";
    }
  } // list boot-error
} // notification sys-startup
```

```
notification sys-config-change {
  description
    "Generated when the <running> configuration is changed.";
  uses sys-common-session-parms;

  list edit {
    description
      "An edit record will be present for each distinct
      edit operation on the running config.";
    leaf target {
      type instance-identifier;
      description
        "Topmost node associated with the configuration change.";
    }

    leaf operation {
      type edit-operation-type;
      description "Type of edit operation performed.";
    }
  } // list edit
} // notification sys-config-change
```

```
notification sys-capability-change {
  description
    "Generated when a <capability> is added, deleted,
    or modified.";
  container changed-by {
    description
      "Indicates who caused this capability change.
      If caused by internal action, then the
      empty leaf 'server' will be present.
      If caused by a management session, then
      the name, remote host address, and session ID
      of the session that made the change will be reported.";
    choice server-or-user {
      leaf server {
        type empty;
        description
          "If present, the capability change was caused
          by the server.";
      }

      case by-user {
        uses sys-common-session-parms;
      } // case by-user
    } // choice server-or-user
  } // container changed-by
```

```
leaf-list added-capability {
  type inet:uri;
  description
    "List of capabilities that have just been added.";
}

leaf-list deleted-capability {
  type inet:uri;
  description
    "List of capabilities that have just been deleted.";
}

leaf-list modified-capability {
  type inet:uri;
  description
    "List of capabilities that have just been modified.";
}
} // notification sys-capability-change

notification sys-session-start {
  description
    "Generated when a new NETCONF session is started.";
  uses sys-common-session-parms;
} // notification sys-session-start

notification sys-session-end {
  description
    "Generated when a NETCONF session is terminated.";
  uses sys-common-session-parms;

  leaf killed-by {
    when "../termination-reason = 'killed'";
    type uint32; // nc:session-id-type;
    description
      "Session ID that issued the <kill-session>
      if the session was terminated by this operation.";
  }

  leaf termination-reason {
    type enumeration {
      enum "closed" {
        value 0;
        description
          "The session was terminated with
          the <close-session> operation.";
      }
    }
  }
}
```

```
enum "killed" {
    value 1;
    description
        "The session was terminated with
        the <kill-session> operation.";
}
enum "dropped" {
    value 2;
    description
        "The session was terminated because
        the SSH session or TCP connection was
        unexpectedly closed.";
}
enum "timeout" {
    value 3;
    description
        "The session was terminated because
        of inactivity, either waiting for
        the <hello> or <rpc> messages.";
}
enum "bad-start" {
    value 4;
    description "The session startup sequence failed.";
}
enum "bad-hello" {
    value 5;
    description
        "The client's <hello> message was
        bad or never arrived.";
}
enum "other" {
    value 6;
    description
        "The session was terminated for
        some other reason.";
}
}
mandatory "true";
description "Reason the session was terminated.";
}
} // notification sys-session-end

notification sys-confirmed-commit {
    description
        "Generated when a confirmed-commit event occurs.";
    uses sys-common-session-parms;
```

```
leaf confirm-event {
  description
    "Indicates the event that caused the notification.";
  type enumeration {
    enum "start" {
      value 0;
      description
        "The confirm-commit procedure has started.";
    }
    enum "cancel" {
      value 1;
      description
        "The confirm-commit procedure has been canceled,
        due to the session being terminated.";
    }
    enum "timeout" {
      value 2;
      description
        "The confirm-commit procedure has been canceled,
        due to the confirm-timeout interval expiring.
        The common session parameters will not be present
        in this sub-mode.";
    }
    enum "extend" {
      value 3;
      description
        "The confirm-commit timeout has been extended.";
    }
    enum "complete" {
      value 4;
      description
        "The confirm-commit procedure has been completed.";
    }
  }
  mandatory "true";
} // notification sys-confirmed-commit
}
```

<CODE ENDS>

3. IANA Considerations

TBD

4. Security Considerations

This document defines a YANG module for reporting of particular system information and system events. Although unlikely, it is possible that data obtained from this module could be used in an attack of some kind, although no specific information in this module is considered sensitive.

TBD: follow Security Consideration guidelines from new template text.

5. Acknowledgements

Some data node definitions in this document are based on information provided by the unix 'uname' program (origin unknown).

This module is based on the yuma-system.yang module, which can be found at: <http://www.netconfcentral.org/modules/yuma-system>.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [I-D.ietf-netmod-yang]
Bjorklund, M., "YANG - A data modeling language for the Network Configuration Protocol (NETCONF)",
draft-ietf-netmod-yang-13 (work in progress), June 2010.
- [I-D.ietf-netmod-yang-types]
Schoenwaelder, J., "Common YANG Data Types",
draft-ietf-netmod-yang-types-09 (work in progress),
April 2010.

6.2. Informative References

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

A.1. 00

Initial version.

Author's Address

Andy Bierman
InterWorking Labs
Scotts Valley, CA
USA

Phone: +1 831 460 7010
Email: andyb@iwl.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2011

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
October 18, 2010

snmp cfg
draft-bjorklund-netmod-snmp-cfg-00

Abstract

This document defines a collection of YANG definitions for configuring SNMP engines.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 2. Keywords | 4 |
| 3. Overview | 5 |
| 4. snmp | 6 |
| 5. snmp-common | 7 |
| 6. snmp-agent | 11 |
| 7. snmp-community | 14 |
| 8. snmp-notification | 16 |
| 9. snmp-target | 19 |
| 10. snmp-target-params | 22 |
| 11. snmp-usm | 24 |
| 12. snmp-vacm | 27 |
| 13. IANA Considerations | 32 |
| 14. Security Considerations | 33 |
| 15. Normative References | 34 |
| Appendix A. Example configurations | 35 |
| Authors' Addresses | 36 |

1. Introduction

TBD.

2. Keywords

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

3. Overview

TBD.

4. snmp

```
<CODE BEGINS> file "snmp.yang"
```

```
module snmp {
  namespace "http://yang-central.org/ns/snmp";
  prefix "snmp";

  include snmp-common {
    revision-date 2010-10-17;
  }
  include snmp-agent {
    revision-date 2010-10-17;
  }
  include snmp-community {
    revision-date 2010-10-17;
  }
  include snmp-notification {
    revision-date 2010-10-17;
  }
  include snmp-target {
    revision-date 2010-10-17;
  }
  include snmp-target-params {
    revision-date 2010-10-17;
  }
  include snmp-vacm {
    revision-date 2010-10-17;
  }
  include snmp-usm {
    revision-date 2010-10-17;
  }

  description
    "This module contains a collection of YANG definitions for
    configuring SNMP engines.";

  revision 2010-10-17 {
    description
      "Initial revision.";
  }
}

<CODE ENDS>
```

5. snmp-common

<CODE BEGINS> file "snmp-common.yang"

```
submodule snmp-common {  
    belongs-to snmp {  
        prefix snmp;  
    }  
  
    description  
        "This submodule contains a collection of common YANG definitions  
        for configuring SNMP engines.";  
  
    revision 2010-10-17 {  
        description  
            "Initial revision.";  
    }  
  
    /* Collection of SNMP features */  
  
    feature proxy {  
        description  
            "A server implements this feature if it can act as an  
            SNMP Proxy";  
    }  
  
    feature multiple-contexts {  
        description  
            "A server implements this feature if it supports other contexts  
            than the default context.";  
    }  
  
    feature notification-filter {  
        description  
            "A server implements this feature if it supports SNMP  
            notification filtering.";  
    }  
  
    /* Collection of SNMP specific data types */  
  
    typedef admin-string {  
        type string {  
            length "0..255";  
        }  
        description  
            "Represents and SnmpAdminString as defined in RFC 3411.";  
        reference
```



```
        "RFC 3411: An Architecture for Describing SNMP Management
          Frameworks";
    }

    typedef identifier {
        type admin-string {
            length "1..32";
        }
        description
            "Identifiers are used to name items in the SNMP configuration
            data store.";
    }

    typedef context-name {
        type admin-string {
            length "0..32";
        }
        description
            "The context type represents an SNMP context name.";
    }

    typedef sec-name {
        type admin-string;
        description
            "The sec-name type represents an SNMP security name.";
    }

    typedef mp-model {
        type union {
            type enumeration {
                enum any { value 0; }
                enum v1  { value 1; }
                enum v2c { value 2; }
                enum v3  { value 3; }
            }
            type int32 {
                range "0..2147483647";
            }
        }
        reference
            "RFC3411: An Architecture for Describing SNMP Management
            Frameworks";
    }

    typedef sec-model {
        type union {
            type enumeration {
                enum v1 { value 1; }
            }
        }
    }
```

```
        enum v2c { value 2; }
        enum usm { value 3; }
    }
    type int32 {
        range "1..2147483647";
    }
}
reference
    "RFC3411: An Architecture for Describing SNMP Management
    Frameworks";
}

typedef sec-model-or-any {
    type union {
        type enumeration {
            enum any { value 0; }
        }
        type sec-model;
    }
    reference
        "RFC3411: An Architecture for Describing SNMP Management
        Frameworks";
}

typedef sec-level {
    type enumeration {
        enum no-auth-no-priv { value 1; }
        enum auth-no-priv    { value 2; }
        enum auth-priv       { value 3; }
    }
    reference
        "RFC3411: An Architecture for Describing SNMP Management
        Frameworks";
}

typedef engine-id {
    type string {
        pattern '([0-9a-fA-F]){2}(:([0-9a-fA-F]){2}){4,31})?';
    }
    description
        "The Engine ID specified as a list of colon-specified hexa-
        decimal octets e.g. '4F:4C:41:71'.";
    reference
        "RFC3411: An Architecture for Describing SNMP Management
        Frameworks";
}

typedef wildcard-object-identifier {
```

```
    type string;
    description
        "The wildcard-object-identifier type represents an SNMP object
        identifier where subidentifiers can be given either as a label,
        in numeric form, or a wildcard, represented by a *.";
}

container snmp {
    description
        "Top-level container for SNMP related configuration and
        status objects.";
}
}

<CODE ENDS>
```

6. snmp-agent

```
<CODE BEGINS> file "snmp-agent.yang"

submodule snmp-agent {

  belongs-to snmp {
    prefix snmp;
  }

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

  include snmp-common;

  revision 2010-10-17 {
    description
      "Initial revision.";
  }

  augment /snmp:snmp {

    container agent {

      description
        "Configuration of the SNMP agent";

      leaf enabled {
        type boolean;
        default "false";
        description
          "Enables the SNMP agent.";
      }

      // FIXME: support multiple endpoints

      leaf ip {
        type inet:ip-address;
        default "0.0.0.0";
        description
          "The IPv4 or IPv6 address to which the agent listens.";
      }

      leaf udp-port {
```

```
    type inet:port-number;
    default "161";
    description
        "The UDP port to which the agent listens.";
}

container version {
    description
        "SNMP version used by the agent";
    leaf v1 {
        type empty;
    }
    leaf v2c {
        type empty;
    }
    leaf v3 {
        type empty;
        must "../engine-id" {
            error-message
                "when v3 is configured, an engine-id must be set";
        }
    }
}

container engine-id {
    presence "Sets the local engine-id.";

    description
        "The local SNMP engine's administratively-assigned unique
        identifier.";
    reference "SNMP-FRAMEWORK-MIB.snmpEngineID";

    leaf enterprise-number {
        type uint32;
        mandatory true;
    }
    choice method {
        mandatory true;
        leaf from-ip {
            type inet:ip-address;
        }
        leaf from-mac-address {
            type yang:mac-address;
        }
        leaf from-text {
            type string {
                length 1..27;
            }
        }
    }
}
```

```
    }
    leaf other {
      type string {
        pattern "[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){0,27}";
      }
    }
  }
}

container system {

  description
    "System group configuration.";

  leaf contact {
    type admin-string;
    default "";
    reference "SNMPv2-MIB.sysContact";
  }

  leaf name {
    type admin-string;
    default "";
    reference "SNMPv2-MIB.sysName";
  }

  leaf location {
    type admin-string;
    default "";
    reference "SNMPv2-MIB.sysLocation";
  }

}

}
}
```

<CODE ENDS>

7. snmp-community

```
<CODE BEGINS> file "snmp-community.yang"

submodule snmp-community {

  belongs-to snmp {
    prefix snmp;
  }

  include snmp-common;
  include snmp-target;

  reference
    "RFC3584: Coexistence between Version 1, Version 2, and Version 3
      of the Internet-standard Network Management Framework";

  revision 2010-10-17 {
    description
      "Initial revision.";
  }

  augment /snmp:snmp {

    list community {
      key index;

      description
        "List of communities";
      reference "SNMP-COMMUNITY-MIB.snmpCommunityTable";

      leaf index {
        type snmp:identifier;
        description "Index into the community list.";
        reference "SNMP-COMMUNITY-MIB.snmpCommunityIndex";
      }
      leaf name {
        type string;
        description
          "Use only when the community string is not the same as the
            index.";
        reference "SNMP-COMMUNITY-MIB.snmpCommunityName";
      }
      leaf sec-name {
        type snmp:sec-name;
        description
          "If not set, the value of 'name' is operationally used";
        reference "SNMP-COMMUNITY-MIB.snmpCommunitySecurityName";
      }
    }
  }
}
```

```
    }
    leaf engine-id {
      if-feature snmp:proxy;
      type snmp:engine-id;
      description
        "If not set, the value of the local SNMP engine is
        operationally used by the device.";
      reference "SNMP-COMMUNITY-MIB.snmpCommunityContextEngineID";
    }
    leaf context {
      if-feature snmp:multiple-contexts;
      type snmp:context-name;
      default "";
      reference "SNMP-COMMUNITY-MIB.snmpCommunityContextName";
    }
    leaf target-tag {
      type leafref {
        path "/snmp/target/tag";
      }
      description
        "Used to limit access for this community to the specified
        targets.";
      reference "SNMP-COMMUNITY-MIB.snmpCommunityTransportTag";
    }
  }
}
```

<CODE ENDS>

8. snmp-notification

```
<CODE BEGINS> file "snmp-notification.yang"

submodule snmp-notification {

  belongs-to snmp {
    prefix snmp;
  }

  include snmp-common;
  include snmp-target;
  include snmp-target-params;

  reference
    "RFC3413: Simple Network Management Protocol (SNMP) Applications
    SNMP-NOTIFICATION-MIB";

  revision 2010-10-17 {
    description
      "Initial revision.";
  }

  augment /snmp:snmp/snmp:target {
    leaf notify-profile {
      if-feature snmp:notification-filter;
      type leafref {
        path "/snmp/notify-profile/name";
      }
    }
  }

  augment /snmp:snmp {
    list notify {

      key name;

      description
        "Targets that will receive notifications.";
      reference "SNMP-NOTIFY-MIB.snmpNotifyTable";

      leaf name {
        type snmp:identifier;
        description
          "An arbitrary name for the list entry.";
        reference "SNMP-NOTIFY-MIB.snmpNotifyName";
      }
    }
  }
}
```

```
leaf tag {
  type leafref {
    path "/snmp/target/tag";
  }
  mandatory true;
  description
    "Target tag, selects a set of notification targets.";
  reference "SNMP-NOTIFY-MIB.snmpNotifyTag";
}
leaf type {
  type enumeration {
    enum trap { value 1; }
    enum inform { value 2; }
  }
  must
    '. != inform or '
    + 'not(/snmp/target[tag = current()/../name] '
    + '      ../usm[../engine-id] != '
    + '      /snmp/target[tag = current()/../name]/../usm)' {
    error-message
      "When inform is configured, all v3 targets must have an
      engine-id configured.";
    }
  default trap;
  description "Defines the notification type to be generated.";
  reference "SNMP-NOTIFY-MIB.snmpNotifyType";
}
}

list notify-profile {
  if-feature snmp:notification-filter;
  key name;

  description
    "Notification filter profiles associated with targets.";
  reference "SNMP-NOTIFY-MIB.snmpNotifyFilterProfileTable";

  leaf name {
    type snmp:identifier;
    description "Name of the filter profile";
    reference "SNMP-NOTIFY-MIB.snmpNotifyFilterProfileName";
  }
  list subtree {
    key "oids";

    reference "SNMP-NOTIFY-MIB.snmpNotifyFilterTable";

    leaf oids {
```

```
    type wildcard-object-identifier;
    description
        "A family of subtrees included in this filter.";
    reference "SNMP-NOTIFY-MIB.snmpNotifyFilterSubtree
              SNMP-NOTIFY-MIB.snmpNotifyFilterMask";
}

choice type {
    mandatory true;
    leaf included {
        type empty;
        description
            "The family of subtrees is included in the filter.";
    }
    leaf excluded {
        type empty;
        description
            "The family of subtrees is excluded from the filter.";
    }
    reference "SNMP-NOTIFY-MIB.snmpNotifyFilterType";
}
}
}
}
}
```

<CODE ENDS>

9. snmp-target

<CODE BEGINS> file "snmp-target.yang"

```
submodule snmp-target {  
    belongs-to snmp {  
        prefix snmp;  
    }  
  
    import ietf-inet-types {  
        prefix inet;  
    }  
  
    include snmp-common;  
    include snmp-usm;  
  
    reference  
        "RFC3413: Simple Network Management Protocol (SNMP) Applications  
        SNMP-TARGET-MIB";  
  
    revision 2010-10-17 {  
        description  
            "Initial revision.";  
    }  
  
    augment /snmp:snmp {  
  
        list target {  
            key name;  
  
            description "List of targets.";  
            reference "SNMP-TARGET-MIB.snmpTargetAddrTable";  
  
            leaf name {  
                type snmp:identifier;  
                description  
                    "Identifies the target.";  
                reference "SNMP-TARGET-MIB.snmpTargetAddrName";  
            }  
  
            // make a choice here so we can add other transports, or  
            // they can augment.  
  
            leaf ip {  
                type inet:ip-address;  
                mandatory true;  
                description "Transport IP address of the target";  
            }  
        }  
    }  
}
```

```
        reference "SNMP-TARGET-MIB.snmpTargetAddrTDomain
                SNMP-TARGET-MIB.snmpTargetAddrTAddress";
    }
    leaf udp-port {
        type inet:port-number;
        default 162;
        description "UDP port number";
        reference "SNMP-TARGET-MIB.snmpTargetAddrTDomain
                SNMP-TARGET-MIB.snmpTargetAddrTAddress";
    }
    leaf-list tag {
        type snmp:identifier;
        description
            "List of tag values used to select target address.";
        reference "SNMP-TARGET-MIB.snmpTargetAddrTagList";
    }

    leaf timeout {
        type uint32;
        units "0.01 seconds";
        default 1500;
        description
            "Needed only if this target can receive v3 informs.";
        reference "SNMP-TARGET-MIB.snmpTargetAddrTimeout";
    }
    leaf retries {
        type uint8;
        default 3;
        description
            "Needed only if this target can receive v3 informs.";
        reference "SNMP-TARGET-MIB.snmpTargetAddrRetryCount";
    }
    leaf engine-id {
        type leafref {
            path "/snmp/usm/remote/engine-id";
        }
        must '../usm/user-name' {
            error-message
                "When engine-id is set, usm/user-name must also be set.";
        }
        must '/snmp/usm/remote[engine-id=current()]/'
            + 'user[name=current()../usm/user-name]' {
            error-message
                "When engine-id is set, the usm/user-name must exist in the
                /snmp/usm/remote list for this engine-id.";
        }
        description
            "Needed only if this target can receive v3 informs.
```

```
        This object is not present in the SNMP MIBs.  In
        RFC 3412, it is a implementation specific matter how this
        engine-id is handled.";
        reference "RFC 3412 7.1.9a";
    }
}
}
}

<CODE ENDS>
```

10. snmp-target-params

```
<CODE BEGINS> file "snmp-target-params.yang"

submodule snmp-target-params {

  belongs-to snmp {
    prefix snmp;
  }

  include snmp-common;
  include snmp-community;
  include snmp-target;

  reference
    "RFC3413: Simple Network Management Protocol (SNMP) Applications
    SNMP-TARGET-MIB";

  revision 2010-10-17 {
    description
      "Initial revision.";
  }

  augment /snmp:snmp/snmp:target {

    /* By including the params directly in the target entry we
       lose some flexibility, but we get a simpler model with less
       cross-references. In SNMP, two addrEntries can point to the
       same paramsEntry.
    */
    choice params {
      mandatory true;
      reference "SNMP-TARGET-MIB.snmpTargetParamsTable";
      container v1 {
        description "SNMPv1 parameters type";
        // mp-model is v1, sec-level is noAuthNoPriv
        leaf community {
          type leafref {
            path "/snmp/community/index";
          }
          mandatory true;
          reference "SNMP-TARGET-MIB.snmpTargetParamsSecurityName";
        }
      }
      container v2c {
        description "SNMPv2 community parameters type";
        // mp-model is v2c, sec-level is noAuthNoPriv
        leaf community {
```

```

    type leafref {
      path "/snmp/community/index";
    }
    mandatory true;
    reference "SNMP-TARGET-MIB.snmpTargetParamsSecurityName";
  }
}
container usm {
  description "User based SNMPv3 parameters type";
  // mp-model is v3
  leaf user-name {
    type leafref {
      path "/snmp/usm/local/user/name";
    }
    mandatory true;
    reference "SNMP-TARGET-MIB.snmpTargetParamsSecurityName";
  }
  leaf sec-level {
    type sec-level;
    mandatory true;
    reference "SNMP-TARGET-MIB.snmpTargetParamsSecurityLevel";
  }
}
}
}
}
<CODE ENDS>
```


11. snmp-usm

<CODE BEGINS> file "snmp-usm.yang"

```
submodule snmp-usm {

  belongs-to snmp {
    prefix snmp;
  }

  include snmp-common;

  description
    "This submodule contains a collection of YANG definitions for
    configuring the User-based Security Model (USM) of SNMP.";
  reference
    "RFC3414: User-based Security Model (USM) for version 3 of the
    Simple Network Management Protocol (SNMPv3).";

  revision 2010-10-17 {
    description
      "Initial revision.";
  }

  grouping key {
    choice key-type {
      leaf password {
        /* This must be stored in the config; it cannot be derived from
        the SNMP table. Also, if SNMP is used to set the key,
        this password will not be used anymore */
        type string;
        description
          "Will be used to create a localized key.";
      }
      leaf key {
        type string {
          pattern '([0-9a-fA-F]){2}(:([0-9a-fA-F]){2})*';
        }
        description
          "Authentication key specified as a list of colon-specified
          hexa-decimal octets";
      }
    }
  }

  grouping user-list {
    list user {
      key "name";
    }
  }
}
```

```
reference "SNMP-USER-BASED-SM-MIB.usmUserTable";

leaf name {
  type snmp:identifier;
  reference "SNMP-USER-BASED-SM-MIB.usmUserName
            SNMP-USER-BASED-SM-MIB.usmUserSecurityName";
}
leaf security-name {
  type snmp:identifier;
  description
    "If not set, the value of 'name' is operationally used";
  reference "SNMP-USER-BASED-SM-MIB.usmUserSecurityName";
}
container auth {
  presence "enables authentication";
  description "Enables authentication protocol of the user";
  choice protocol {
    mandatory true;
    reference "SNMP-USER-BASED-SM-MIB.usmUserAuthProtocol";
    container md5 {
      presence "md5";
      uses key;
    }
    container sha {
      presence "sha";
      uses key;
    }
  }
}
container priv {
  must "../auth" {
    error-message
      "when privacy is used, authentication must also be used";
  }
  presence "enables encryption";
  description
    "Enables encryption for the authentication process.";

  choice protocol {
    mandatory true;
    reference "SNMP-USER-BASED-SM-MIB.usmUserPrivProtocol";
    container des {
      presence "des";
      uses key;
    }
    container aes {
      presence "aes";
      uses key;
    }
  }
}
```

```
    }
  }
}

augment /snmp:snmp {

  container usm {
    description
      "Configuration of the User-based Security Model";
    container local {
      uses user-list;
    }

    list remote {
      key "engine-id";

      leaf engine-id {
        type snmp:engine-id;
        reference "SNMP-USER-BASED-SM-MIB.usmUserEngineID";
      }

      uses user-list;
    }
  }
}

<CODE ENDS>
```

12. snmp-vacm

<CODE BEGINS> file "snmp-vacm.yang"

```
submodule snmp-vacm {  
    belongs-to snmp {  
        prefix snmp;  
    }  
  
    include snmp-common;  
  
    description  
        "This submodule contains a collection of YANG definitions for  
        configuring the View-based Access Control Model (VACM) of SNMP.";  
    reference  
        "RFC3415: View-based Access Control Model (VACM) for the  
        Simple Network Management Protocol (SNMP)";  
  
    revision 2010-10-17 {  
        description  
            "Initial revision.";  
    }  
  
    typedef view-name {  
        type snmp:identifier;  
        description  
            "The view-name type represents an SNMP VACM view name.";  
    }  
  
    typedef group-name {  
        type snmp:identifier;  
        description  
            "The group-name type represents an SNMP VACM group name.";  
    }  
  
    augment /snmp:snmp {  
        container vacm {  
            description  
                "Configuration of the View-based Access Control Model";  
  
            list group {  
                key name;  
                description  
                    "VACM Groups";  
                reference "SNMP-VIEW-BASED-ACM-MIB.vacmSecurityToGroupTable";  
            }  
        }  
    }  
}
```

```
leaf name {
    type group-name;
    description
        "The name of this VACM group.";
    reference "SNMP-VIEW-BASED-ACM-MIB.vacmGroupName";
}

list member {
    key "sec-name";
    min-elements 1;
    description
        "A member of this VACM group. According to VACM, every
        group must have at least one member.

        A certain combination of sec-name and sec-model MUST NOT
        be mapped to more than one group.";

    leaf sec-name {
        type snmp:sec-name;
        description
            "The securityName of a group member.";
    }

    leaf-list sec-model {
        type snmp:sec-model;
        min-elements 1;
        description
            "The security models under which this securityName
            is a member of this group.";
    }
}

list access {
    key "context sec-model sec-level";
    description
        "Definition of access right for groups";
    reference "SNMP-VIEW-BASED-ACM-MIB.vacmAccessTable";

    leaf context {
// FIXME: since this is part of the key, it must not have an if-feature
//         if-feature snmp:multiple-contexts;
        type snmp:context-name;
        description
            "The context (prefix) under which the access rights
            apply.";
        reference
            "SNMP-VIEW-BASED-ACM-MIB.vacmAccessContextPrefix";
    }
}
```

```
leaf context-match {
  if-feature snmp:multiple-contexts;
  type enumeration {
    enum exact;
    enum prefix;
  }
  default exact;
  reference
    "SNMP-VIEW-BASED-ACM-MIB.vacmAccessContextMatch";
}

leaf sec-model {
  type snmp:sec-model-or-any;
  description
    "The security model under which the access rights
    apply.";
  reference
    "SNMP-VIEW-BASED-ACM-MIB.vacmAccessSecurityModel";
}

leaf sec-level {
  type snmp:sec-level;
  description
    "The minimum security level under which the access rights
    apply.";
  reference
    "SNMP-VIEW-BASED-ACM-MIB.vacmAccessSecurityLevel";
}

leaf read-view {
  type leafref {
    path "/snmp/vacm/view/name";
  }
  description
    "The name of the MIB view of the SNMP context authorizing
    read access.";
  reference
    "SNMP-VIEW-BASED-ACM-MIB.vacmAccessReadViewName";
}

leaf write-view {
  type leafref {
    path "/snmp/vacm/view/name";
  }
  description
    "The name of the MIB view of the SNMP context authorizing
    write access.";
  reference
```

```
        "SNMP-VIEW-BASED-ACM-MIB.vacmAccessWriteViewName";
    }

    leaf notify-view {
        type leafref {
            path "/snmp/vacm/view/name";
        }
        description
            "The name of the MIB view of the SNMP context authorizing
            notify access.";
        reference
            "SNMP-VIEW-BASED-ACM-MIB.vacmAccessNotifyViewName";
    }
}

list view {
    key name;
    description
        "Definition of MIB views";
    reference
        "SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilyTable";

    leaf name {
        type view-name;
        description
            "The name of this VACM MIB view.";
        reference
            "SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilyName";
    }

    list subtree {
        key "oids";
        reference
            "SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilySubtree";

        leaf oids {
            type snmp:wildcard-object-identifier;
            description
                "A family of subtrees included in this MIB view.";
            reference
                "SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilySubtree
                SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilyMask";
        }

        choice type {
            mandatory true;
            reference "SNMP-VIEW-BASED-ACM-MIB.vacmViewTreeFamilyType";
        }
    }
}
```

```
        leaf included {
            type empty;
            description
                "The family of subtrees is included in the MIB view";
        }
        leaf excluded {
            type empty;
            description
                "The family of subtrees is excluded from the MIB view";
        }
    }
}
}
```

<CODE ENDS>

13. IANA Considerations

TBD.

14. Security Considerations

TBD.

15. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Appendix A. Example configurations

TBD.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

NETCONF Data Modeling Language
Internet-Draft
Intended status: Informational
Expires: April 21, 2011

Q. Chen
M. Du
ZTE Corporation
Oct 18, 2010

Extending YANG with Revised Types
draft-chen-netmod-yang-ext-00

Abstract

YANG - the NETCONF Data Modeling Language - supports modeling of a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF. This document introduces new idea which revises the ID [draft-linowski-netmod-yang-abstract-03] and clears some ambiguous concepts and descriptions.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 3 |
| 1.1. Conventions used in this document | 3 |
| 2. Redefining type in YANG abstract | 3 |
| 3. Revised types in yang-abstract | 3 |
| 3.1. Removing confusion | 3 |
| 3.2. Result | 4 |
| 4. Management Consideration | 4 |
| 5. Security Considerations | 4 |
| 6. IANA Considerations | 4 |
| Authors' Addresses | 4 |

1. Introduction

[draft-linowski-netmod-yang-abstract-03] suggests to enhance YANG with supplementary modeling features and language abstractions with the aim to improve the model extensibility and reuse.

However, some ideas and description are not defined clearly, this memo tries to amend the concepts and gives some suggestions.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Redefining type in YANG abstract

section "2.3. instance extension statement" and "2.4. instance-list extension statement", the table of substatement list out that "type" is allowed, but "instance-type" is not listed. so it is conflict with what is wrote in section 3.2. instance-type extension statement. suggest that "instance-type" is need not defined, use "type" instead, and give description in section 2.3 and 2.4

3. Revised types in yang-abstract

3.1. Removing confusion

There exists difference between types of a list and types of the node of the list, and both types are being used in application implementation. But in YANG and yang-abstract, it is not distinguished. It will cause confusion in comprehension and implementation. So it is strongly suggested do not introduce subtree/subtype/field in list definition, that is in "draft-linowski-netmod-yang-abstract-03.txt", section "2.4. instance-list extension statement" remove choice, container, instance, instance-list, leaf, leaf-list, list in substatement table, only "type" for use a complex-type is allowed.

3.2. Result

| substatement | cardinality |
|--------------|-------------|
| description | 0..1 |
| config | 0..1 |
| if-feature | 0..n |
| mandatory | 0..1 |
| must | 0..n |
| reference | 0..1 |
| status | 0..1 |
| type | 1 |

Figure 1: instance's substatements

4. Management Consideration

5. Security Considerations

TBD

6. IANA Considerations

TBD

Authors' Addresses

Qiaogang Chen
ZTE Corporation
3/F, R.D. Building 3, ZTE Industrial Park, Liuxian Road
Shenzhen 518055
P.R.China

Phone: +86 755 26773712
Email: chen.qiaogang@zte.com.cn
URI: <http://www.zte.com.cn/>

Ming Du
ZTE Corporation
3/F, R.D. Building 3, ZTE Industrial Park, Liuxian Road
Shenzhen 518055
P.R.China

Phone: +86 755 26773712
Email: du.ming@zte.com.cn
URI: <http://www.zte.com.cn/>

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: May 15, 2011

A. Bierman
Brocade
B. Lengyel
Ericsson
November 11, 2010

With-defaults capability for NETCONF
draft-ietf-netconf-with-defaults-14

Abstract

The NETCONF protocol defines ways to read and edit configuration data from a NETCONF server. In some cases, part of this data may not be set by the NETCONF client, but rather a default value known to the server is used instead. In many situations the NETCONF client has a priori knowledge about default data, so the NETCONF server does not need to save it in a NETCONF configuration datastore or send it to the client in a retrieval operation reply. In other situations the NETCONF client will need this data from the server. Not all server implementations treat this default data the same way. This document defines a capability-based extension to the NETCONF protocol that allows the NETCONF client to identify how defaults are processed by the server, and also defines new mechanisms for client control of server processing of default data.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 15, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 4 |
| 1.1. Terminology | 4 |
| 1.2. Defaults Handling Behavior | 5 |
| 1.3. Client Controlled Retrieval of Default Data | 5 |
| 2. Defaults Handling Basic Modes | 6 |
| 2.1. 'report-all' Basic Mode | 6 |
| 2.1.1. 'report-all' Basic Mode Retrieval | 7 |
| 2.1.2. 'report-all' <with-defaults> Retrieval | 7 |
| 2.1.3. 'report-all' <edit-config> and <copy-config> Behavior | 7 |
| 2.2. 'trim' Basic Mode | 7 |
| 2.2.1. 'trim' Basic Mode Retrieval | 7 |
| 2.2.2. 'trim' <with-defaults> Retrieval | 7 |
| 2.2.3. 'trim' <edit-config> and <copy-config> Behavior | 8 |
| 2.3. 'explicit' Basic Mode | 8 |
| 2.3.1. 'explicit' Basic Mode Retrieval | 8 |
| 2.3.2. 'explicit' <with-defaults> Retrieval | 8 |
| 2.3.3. 'explicit' <edit-config> and <copy-config> Behavior | 8 |
| 3. Retrieval of Default Data | 9 |
| 3.1. 'report-all' Retrieval Mode | 9 |
| 3.2. 'trim' Retrieval Mode | 9 |
| 3.3. 'explicit' Retrieval Mode | 9 |
| 3.4. 'report-all-tagged' Retrieval Mode | 10 |
| 4. With-defaults Capability | 10 |
| 4.1. Overview | 10 |
| 4.2. Dependencies | 11 |
| 4.3. Capability Identifier | 11 |
| 4.4. New Operations | 11 |
| 4.5. Modifications to Existing Operations | 11 |
| 4.5.1. <get>, <get-config>, and <copy-config> Operations | 11 |
| 4.5.2. <edit-config> Operation | 13 |
| 4.5.3. Other Operations | 13 |
| 4.6. Interactions with Other Capabilities | 14 |
| 5. YANG Module for the <with-defaults> Parameter | 14 |
| 6. XSD for the 'default' Attribute | 17 |

| | |
|--|----|
| 7. IANA Considerations | 19 |
| 8. Security Considerations | 19 |
| 9. Acknowledgements | 19 |
| 10. Normative References | 20 |
| Appendix A. Usage Examples | 20 |
| A.1. Example YANG Module | 20 |
| A.2. Example Data Set | 22 |
| A.3. Protocol Operation Examples | 23 |
| A.3.1. <with-defaults> = 'report-all' | 23 |
| A.3.2. <with-defaults> = 'report-all-tagged' | 24 |
| A.3.3. <with-defaults> = 'trim' | 27 |
| A.3.4. <with-defaults> = 'explicit' | 28 |
| Appendix B. Change Log | 29 |
| B.1. 13-14 | 29 |
| B.2. 12-13 | 29 |
| B.3. 11-12 | 29 |
| B.4. 10-11 | 29 |
| B.5. 09-10 | 29 |
| B.6. 08-09 | 29 |
| B.7. 07-08 | 30 |
| B.8. 06-07 | 30 |
| B.9. 05-06 | 30 |
| B.10. 04-05 | 31 |
| B.11. 03-04 | 31 |
| B.12. 02-03 | 31 |
| B.13. 01-02 | 32 |
| B.14. 00-01 | 32 |
| B.15. -00 | 32 |
| Authors' Addresses | 32 |

1. Introduction

The NETCONF protocol [I-D.ietf-netconf-4741bis] defines ways to read configuration and state data from a NETCONF server. Part of the configuration data may not be set by the NETCONF client, but rather by a default value from the data model. In many situations the NETCONF client has a priori knowledge about default data, so the NETCONF server does not need to send it to the client. A priori knowledge can be obtained, e.g., a document formally describing the data models supported by the NETCONF server.

It can be important for a client to know exactly how a server implementation will handle default data. There are subtle differences in some protocol operations where the defaults handling behavior of the server will affect the outcome of the operation.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Data model schema: A document or set of documents describing the data models supported by the NETCONF server.

Management Application: A computer program running outside the NETCONF server that configures or supervises the NETCONF server. A management application can reach the device e.g. via NETCONF, command line interface (CLI) or Simple Network Management Protocol (SNMP).

Schema default data: Data specified in the data model schema as default, that is set or used by the device whenever the NETCONF client or other management application/user does not provide a specific value for the relevant data node. Schema default data may or may not be stored as part of a configuration datastore, depending on the basic mode used by a particular server.

Default data: Conceptual data containing a default value. Default data is not kept in a datastore. Not all servers use the same criteria to decide if a data node is actually instantiated in a datastore. If a data node is not present in a datastore, and a schema default definition is in use by the server instead, then it is considered to be a default data node.

Default value: A default value is a value for a data node instance that is conceptually in use by the server, when the data node instance does not exist.

Explicitly set data: Data that is set to any value by a NETCONF client or other management application by the way of an explicit management operation, including any data model schema default value. Any value set by the NETCONF server which is not the schema defined default value is also considered explicitly set data.

<with-defaults> retrieval: Refers to a protocol operation which includes the <with-default> parameter to control the handling of default data.

:with-defaults: The shorthand notation for the with-defaults capability identifier.

The following terms are defined in [I-D.ietf-netconf-4741bis]:

- o client
- o datastore
- o operation
- o server

The following term is defined in [RFC6020]:

- o data node

1.2. Defaults Handling Behavior

The defaults handling behavior used by a server will impact NETCONF protocol operations in two ways:

1. Data retrieval: A server is normally allowed to exclude data nodes which it considers to contain the default value. The actual nodes omitted depends on the defaults handling behavior used by the server.
2. Create and delete operations: The <edit-config> 'operation' attribute can be used to create and/or delete specific data nodes. These operations depend on whether the target node currently exists or not. The server's defaults handling behavior will determine whether the requested node currently exists in the configuration datastore or not.

1.3. Client Controlled Retrieval of Default Data

A networking device may have a large number of default values. Often the default values are specifically defined with a reasonable value, documented and well-known, so that the management user does not need to handle them. For these reasons it is quite common for networking devices to suppress the output of parameters having the default value.

However, there are use-cases when a NETCONF client will need the default data from the server:

- o The management application often needs a single, definitive and complete set of configuration values that determine how the networking device works.
- o Documentation about default values can be unreliable or unavailable.
- o Some management applications might not have the capabilities to correctly parse and interpret formal data models.
- o Human users might want to understand the received data without consultation of the documentation.

In all these cases, the NETCONF client will need a mechanism to retrieve default data from a NETCONF server.

This document defines a NETCONF protocol capability to identify the server defaults handling behavior, an XML attribute to identify default data, and a YANG module extension to the NETCONF protocol that allows the NETCONF client to control whether default data is returned by the server.

2. Defaults Handling Basic Modes

Not all server implementations treat default data in the same way. Instead of forcing a single implementation strategy, this document allows a server to advertise a particular style of defaults handling, and the client can adjust accordingly.

NETCONF servers report default data in different ways. This document specifies three standard defaults handling basic modes that a server implementor may choose from:

- o report-all
- o trim
- o explicit

A server **MUST** select one of the three basic modes defined in this section for handling default data.

2.1. 'report-all' Basic Mode

A server which uses the 'report-all' basic mode does not consider any data node to be default data, even schema default data.

2.1.1. 'report-all' Basic Mode Retrieval

When data is retrieved from a server using the 'report-all' basic mode, and the <with-defaults> parameter is not present, all data nodes MUST be reported.

2.1.2. 'report-all' <with-defaults> Retrieval

If the 'report-all' basic mode is used by the server, then the server MUST support the <with-defaults> parameter with a value equal to 'report-all', as specified in Section 3.1.

2.1.3. 'report-all' <edit-config> and <copy-config> Behavior

The server MUST consider every data node to exist, even those containing a schema default value. A valid 'create' operation attribute for a data node that contains its schema default value MUST fail with a 'data-exists' error-tag. A valid 'delete' operation attribute for a data node that contains its schema default value MUST succeed, even though the data node is immediately replaced by the server with the default value.

A server which uses the 'report-all' basic-mode has no concept of a default node, so the 'report-all-tagged' <with-defaults> retrieval mode is not relevant. There will never be any tagged nodes, since there are no nodes which are omitted in a basic-mode retrieval operation. If the 'default' attribute is present in any configuration data, the server MUST return an <rpc-error> response with an 'unknown-attribute' error-tag.

2.2. 'trim' Basic Mode

A server which uses the 'trim' basic mode MUST consider any data node set to its schema default value to be default data.

2.2.1. 'trim' Basic Mode Retrieval

When data is retrieved from a server using the 'trim' basic mode, and the <with-defaults> parameter is not present, data nodes MUST NOT be reported if they contain the schema default value. Non-configuration data nodes containing the schema default value MUST NOT be reported.

2.2.2. 'trim' <with-defaults> Retrieval

If the 'trim' basic mode is used by the server, then the server MUST support the <with-defaults> parameter with a value equal to 'trim', as specified in Section 3.2.

2.2.3. 'trim' <edit-config> and <copy-config> Behavior

The server MUST consider any data node that does not contain its schema default value to exist. A valid 'create' operation attribute for a data node that has a schema default value defined MUST succeed. A valid 'delete' operation attribute for a missing data node that has a schema default value MUST fail. The server MUST return an <rpc-error> response with a 'data-missing' error-tag.

If a client sets a data node to its schema default value, using any valid operation, it MUST succeed, although the data node MUST NOT be saved in the NETCONF configuration datastore. This has the same effect as removing the data node and treating it as default data.

If the server supports the 'report-all-tagged' value for the <with-defaults> parameter, then the 'default' attribute MUST be accepted in configuration input, as described in Section 4.5.1 and Section 4.5.2.

2.3. 'explicit' Basic Mode

A server which uses the 'explicit' basic mode MUST consider any data node that is not explicitly set data to be default data.

2.3.1. 'explicit' Basic Mode Retrieval

When data is retrieved from a server using the 'explicit' basic mode, and the <with-defaults> parameter is not present, data nodes MUST be reported if explicitly set by the client, even if they contain the schema default value. Non-configuration data nodes containing the schema default value MUST be reported.

2.3.2. 'explicit' <with-defaults> Retrieval

If the 'explicit' basic mode is used by the server, the server MUST support the <with-defaults> parameter with a value equal to 'explicit', as specified in Section 3.3.

2.3.3. 'explicit' <edit-config> and <copy-config> Behavior

The server considers any data node that is explicitly set data to exist. A valid 'create' operation attribute for a data node that has been set by a client to its schema default value MUST fail with a 'data-exists' error-tag. A valid 'create' operation attribute for a data node that has been set by the server to its schema default value MUST succeed. A valid 'delete' operation attribute for a data node that has been set by a client to its schema default value MUST succeed. A valid 'delete' operation attribute for a data node that has been set by the server to its schema default value MUST fail with

a 'data-missing' error-tag.

If the server supports the 'report-all-tagged' retrieval mode in its :with-defaults capability, then the 'default' attribute MUST be accepted in configuration input. If all NETCONF <edit-config> or <copy-config> parameters are valid, then the server will treat a tagged data node (i.e., the 'default' attribute set to 'true' or '1') as a request to return that node to default data. If this request is valid within the context of the requested NETCONF operation, then the data node is removed and returned to its default value. The data node within the NETCONF message MUST contain a value in this case, which MUST be equal to the schema default value. If not, the server MUST return an <rpc-error> response with a 'invalid-value' error-tag.

3. Retrieval of Default Data

This document defines a new parameter, called <with-defaults>, which can be added to specific NETCONF operation request messages to control how retrieval of default data is treated by the server.

A server which implements this specification MUST accept the <with-defaults> parameter containing the enumeration for any of the defaults handling modes it supports. The <with-defaults> parameter contains one of the four enumerations defined in this section.

3.1. 'report-all' Retrieval Mode

When data is retrieved with a <with-defaults> parameter equal to 'report-all', all data nodes MUST be reported, including any data nodes considered to be default data by the server.

3.2. 'trim' Retrieval Mode

When data is retrieved with a <with-defaults> parameter equal to 'trim', data nodes MUST NOT be reported if they contain the schema default value. Non-configuration data nodes containing the schema default value MUST NOT be reported.

3.3. 'explicit' Retrieval Mode

When data is retrieved with a <with-defaults> parameter equal to 'explicit', a data node which was set by a client to its schema default value MUST be reported. A conceptual data node which would be set by the server to the schema default value MUST NOT be reported. Non-configuration data nodes containing the schema default value MUST be reported.

3.4. 'report-all-tagged' Retrieval Mode

In addition to the basic modes, a special variant of the 'report-all' basic mode is available called 'report-all-tagged'. This mode MUST be supported on a server if the 'also-supported' parameter in the :with-defaults capability contains the 'report-all-tagged' option. Refer to Section 4 for encoding details for this capability.

In this mode the server returns all data nodes, just like the 'report-all' mode, except a data node that is considered by the server to contain default data will include an XML attribute to indicate this condition. This is useful for an application to determine which nodes are considered to contain default data by the server, within a single retrieval operation.

A server which supports 'report-all-tagged' MUST also accept the 'default' XML attribute within configuration input to the <edit-config> or <copy-config> operations. Refer to Section 6 for XML encoding details of the 'default' XML attribute.

4. With-defaults Capability

4.1. Overview

The :with-defaults capability indicates which defaults handling basic mode is supported by the server. It may also indicate support for additional defaults retrieval modes. These retrieval modes allow a NETCONF client to control whether default data is returned by the server. The capability affects both configuration and state data (while acknowledging that the usage of default values for state data is less prevalent). Sending of default data is controlled for each individual operation separately.

A NETCONF server implementing the :with-defaults capability:

- o MUST indicate its basic mode behavior by including the 'basic-mode' parameter in the capability URI, as defined in Section 4.3.
- o MUST support the YANG module defined in Section 5 for the defaults handling mode indicated by the 'basic-mode' parameter.
- o SHOULD support the YANG module in Section 5 for the defaults handling mode identified by the 'report-all' or 'report-all-tagged' enumeration value.
- o If the 'report-all-tagged' defaults handling mode is supported, then the 'default' attribute MUST be supported.
- o MAY support the YANG module in Section 5 for additional defaults handling modes.

4.2. Dependencies

None

4.3. Capability Identifier

urn:ietf:params:netconf:capability:with-defaults:1.0

The identifier **MUST** have a parameter: "basic-mode". This indicates how the server will treat default data, as defined in Section 2. The allowed values of this parameter are 'report-all', 'trim', and 'explicit', as defined in Section 2.

The identifier **MAY** have another parameter: "also-supported". This parameter indicates which additional enumeration values (besides the basic-mode enumeration), the server will accept for the <with-defaults> parameter in Section 5. The value of the parameter is a comma separated list of one or more modes that are supported beside the mode indicated in the 'basic-mode' parameter. Possible modes are 'report-all', 'report-all-tagged', 'trim', and 'explicit', as defined in Section 3.

Note that this protocol capability URI is separate from the YANG module capability URI for the YANG module in Section 5. A server which implements this module **MUST** also advertise a YANG module capability URI according to the rules specified in [RFC6020].

Examples:

urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit

urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&also-supported=report-all,report-all-tagged

4.4. New Operations

None

4.5. Modifications to Existing Operations

4.5.1. <get>, <get-config>, and <copy-config> Operations

A new <with-defaults> XML element is added to the input for the <get>, <get-config> and <copy-config> operations. If the <with-defaults> element is present, it controls the reporting of default data. The server **MUST** return default data in the NETCONF <rpc-reply> messages according to the value of this element, if the server

supports the specified retrieval mode.

This parameter only controls these specified retrieval operations, and does not impact any other operations or the non-volatile storage of configuration data.

The `<with-defaults>` element is defined in the XML namespace for the `ietf-netconf-with-defaults.yang` module in Section 5, not the XML namespace for the `<get>`, `<get-config>` and `<copy-config>` operations.

Allowed values of the `with-defaults` element are taken from the `'with-defaults-type'` typedef in Section 5. The allowed values for a particular server are restricted to the values that the server indicates it supports within the `:with-defaults` capability, in the `'basic-mode'` and `'also-supported'` parameters.

If an unsupported value is used, the NETCONF server MUST return an `<rpc-error>` response with an `'invalid-value'` error-tag.

If the `<with-defaults>` element is not present, the server MUST follow its basic mode behavior as indicated by the `:with-defaults` capability identifier's `'basic-mode'` parameter, defined in Section 4.3.

The `<get>` and `<get-config>` operations support a separate filtering mechanism, using the `<filter>` parameter. The defaults filtering is conceptually done before the `<filter>` parameter is processed. For example, if the `<with-defaults>` parameter is equal to `'report-all'`, then the `<filter>` parameter is conceptually applied to all data nodes and all default data.

The `<copy-config>` operation is only affected by the `<with-defaults>` parameter if the target of the operation is specified with the `<url>` parameter. If the target is a NETCONF configuration datastore (i.e., running, candidate or startup), the `<with-defaults>` parameter has no effect. The server MUST use its basic mode when copying data to a NETCONF configuration datastore. If the `<with-defaults>` parameter is present in this case, it MUST be silently ignored by the server.

If the server supports the `'report-all-tagged'` mode, then the `'default'` attribute defined in Section 6 also impacts the `<copy-config>` operation. If the `'default'` attribute is present and set to `'true'` or `'1'`, then the server MUST treat the new data node as a request to return that node to its default value (i.e., remove it from the configuration datastore). The data node within the NETCONF message MUST contain a value in this case, which MUST be equal to the schema default value. If not, the server MUST return an `<rpc-error>` response with a `'invalid-value'` error-tag.

4.5.2. <edit-config> Operation

The <edit-config> operation has several editing modes. The 'create' and 'delete' editing operations are affected by the defaults handling basic mode. The other enumeration values for the NETCONF operation attribute are not affected.

If the operation attribute contains the value 'create', and the data node already exists in the target configuration datastore, then the server MUST return an <rpc-error> response with a 'invalid-value' error-tag.

If the client sets a data node to its schema default value, the server MUST accept the request if it is valid. The server MUST keep or discard the new value based on its defaults handling basic mode. For the 'trim' basic mode, all schema default values are discarded, otherwise a client-provided schema default value is saved in a NETCONF configuration datastore.

If the server supports the 'report-all-tagged' mode, then the 'default' attribute defined in Section 6 also impacts the <edit-config> operation. If the 'default' attribute is present and set to 'true' or '1', then the server MUST treat the new data node as a request to return that node to its default value (i.e., remove it from the configuration datastore). The data node within the NETCONF message MUST contain a value in this case, which MUST be equal to the schema default value. If not, the server MUST return an <rpc-error> response with a 'invalid-value' error-tag.

If the 'default' attribute is present, then the effective operation for the target data node MUST be 'create', 'merge' or 'replace'. If not, then the server MUST return an <rpc-error> response with an 'invalid-value' error-tag. For example, if 'create' is the effective operation, then the create request must be valid on its own (e.g., current data node MUST NOT exist). The procedure for determining the effective operation is defined in [I-D.ietf-netconf-4741bis]. It is derived from the 'default-operation' parameter and/or any operation attributes that are present in the data node or any of its ancestor nodes, within the <edit-config> request.

4.5.3. Other Operations

Other operations that return configuration data SHOULD also handle default data according to the rules set in this document, and explicitly state this in their documentation. If this is not specified in the document defining the respective operation, the default handling rules described herein do not affect these operations.

4.6. Interactions with Other Capabilities

None

5. YANG Module for the <with-defaults> Parameter

The following YANG module defines the addition of the with-defaults element to the <get>, <get-config>, and <copy-config> operations. The YANG language is defined in [RFC6020]. The above operations are defined in YANG in [I-D.ietf-netconf-4741bis]. Every NETCONF server which supports the :with-defaults capability MUST implement this YANG module.

```
<CODE BEGINS> file="ietf-netconf-with-defaults@2010-11-11.yang"

module ietf-netconf-with-defaults {

    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults";

    prefix ncwd;

    import ietf-netconf { prefix nc; }

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

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

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

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

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

        Editor: Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>";

    description
        "This module defines an extension to the NETCONF protocol
        that allows the NETCONF client to control how default
```

values are handled by the server in particular NETCONF operations.

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

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

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

// RFC Ed.: replace XXXX with actual RFC number and remove this note

// RFC Ed.: remove this note

// Note: extracted from draft-ietf-netmod-with-defaults-14.txt

revision 2010-11-11 {

 description

 "Initial version.";

 reference

 "RFC XXXX: With-defaults capability for NETCONF";

}

// RFC Ed.: replace XXXX with actual

// RFC number and remove this note

typedef with-defaults-mode {

 description

 "Possible modes to report default data.";

 reference

 "RFC XXXX; section 3.";

 // RFC Ed.: replace XXXX with actual

 // RFC number and remove this note

 type enumeration {

 enum report-all {

 description

 "All default data is reported.";

 reference

 "RFC XXXX; section 3.1";

 // RFC Ed.: replace XXXX with actual

 // RFC number and remove this note

 }

 enum report-all-tagged {

 description

```
        "All default data is reported.
        Any nodes considered to be default data
        will contain a 'default' XML attribute,
        set to 'true' or '1'.";
    reference
        "RFC XXXX; section 3.4";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note
}
enum trim {
    description
        "Values are not reported if they contain the default.";
    reference
        "RFC XXXX; section 3.2";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note
}
enum explicit {
    description
        "Report values that contain the definition of
        explicitly set data.";
    reference
        "RFC XXXX; section 3.3";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note
}
}
}

grouping with-defaults-parameters {
    description
        "Contains the <with-defaults> parameter for control
        of defaults in NETCONF retrieval operations.";

    leaf with-defaults {
        description
            "The explicit defaults processing mode requested.";
        reference
            "RFC XXXX; section 4.6.1";
            // RFC Ed.: replace XXXX with actual
            // RFC number and remove this note

        type with-defaults-mode;
    }
}

// extending the get-config operation
```

```
augment /nc:get-config/nc:input {
  description
    "Adds the <with-defaults> parameter to the
    input of the NETCONF <get-config> operation.";
  reference
    "RFC XXXX; section 4.6.1";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note

  uses with-defaults-parameters;
}

// extending the get operation
augment /nc:get/nc:input {
  description
    "Adds the <with-defaults> parameter to
    the input of the NETCONF <get> operation.";
  reference
    "RFC XXXX; section 4.6.1";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note

  uses with-defaults-parameters;
}

// extending the copy-config operation
augment /nc:copy-config/nc:input {
  description
    "Adds the <with-defaults> parameter to
    the input of the NETCONF <copy-config> operation.";
  reference
    "RFC XXXX; section 4.6.1";
    // RFC Ed.: replace XXXX with actual
    // RFC number and remove this note

  uses with-defaults-parameters;
}
}

<CODE ENDS>
```

6. XSD for the 'default' Attribute

The following XML Schema document [W3C.REC-xml-20081126] defines the 'default' attribute, described within this document. This XSD is

only relevant if the server supports the 'report-all-tagged' defaults retrieval mode.

The 'default' attribute uses the XSD data type 'boolean'. In accordance with Section 3.2.2.1 of XML Schema Part 2: Datatypes, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept of false and the strings "1" and "true" for the concept of true. Implementations MUST support both styles of lexical representation.

<CODE BEGINS> file="defaults.xsd"

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

  <xs:annotation>
    <xs:documentation>
      This schema defines the syntax for the 'default' attribute
      described within this document.
    </xs:documentation>
  </xs:annotation>

  <!--
    default attribute
  -->
  <xs:attribute name="default" type="xs:boolean" default="false">
    <xs:annotation>
      <xs:documentation>
        This attribute indicates whether the data node represented
        by the XML element containing this attribute is considered
        by the server to be default data. If set to 'true' or '1' then
        the data node is default data. If 'false' or '0', then the
        data node is not default data.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

</xs:schema>

<CODE ENDS>
```

7. IANA Considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol Capability URNs registry':

```
urn:ietf:params:netconf:capability:with-defaults:1.0
```

Note that the capability URN is compliant to [I-D.ietf-netconf-4741bis] section 10.3.

This document registers two XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

```
URI: urn:ietf:params:xml:ns:netconf:default:1.0
```

```
URI: urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults
```

Registrant Contact: The NETCONF WG of the IETF.

XML: N/A, the requested URIs are XML namespaces.

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020] .

```
name: ietf-netconf-with-defaults
```

```
prefix: ncwd
```

```
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults
```

```
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment
```

8. Security Considerations

This document defines an extension to existing NETCONF protocol operations. It does not introduce any new or increased security risks into the management system.

The 'with-defaults' capability gives clients control over the retrieval of default data from a NETCONF datastore. The security consideration of [I-D.ietf-netconf-4741bis] apply to this document as well.

9. Acknowledgements

Thanks to Martin Bjorklund, Sharon Chisholm, Phil Shafer, Juergen Schoenwaelder, Kent Watsen, Washam Fan and many other members of the NETCONF WG for providing important input to this document.

10. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [I-D.ietf-netconf-4741bis]
Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", draft-ietf-netconf-4741bis-06 (work in progress), October 2010.
- [W3C.REC-xml-20081126]
Maler, E., Yergeau, F., Sperberg-McQueen, C., Paoli, J., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [W3C.REC-xmlschema-0-20041028]
Walmsley, P. and D. Fallside, "XML Schema Part 0: Primer Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-0-20041028, October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>>.

Appendix A. Usage Examples

A.1. Example YANG Module

The following YANG module defines an example interfaces table to demonstrate how the <with-defaults> parameter behaves for a specific data model.

Note that this is not a real module, and implementation of this module is not required for conformance to the :with-defaults capability, defined in Section 4. This module is not to be registered with IANA, and is not considered to be a code component. It is intentionally very terse, and includes few descriptive statements.


```
module example {  
  namespace "http://example.com/ns/interfaces";  
  prefix exam;  
  
  typedef status-type {  
    description "Interface status";  
    type enumeration {  
      enum ok;  
      enum 'waking up';  
      enum 'not feeling so good';  
      enum 'better check it out';  
      enum 'better call for help';  
    }  
    default ok;  
  }  
  
  container interfaces {  
    description "Example interfaces group";  
  
    list interface {  
      description "Example interface entry";  
      key name;  
  
      leaf name {  
        description  
          "The administrative name of the interface.  
          This is an identifier which is only unique  
          within the scope of this list, and only  
          within a specific server.";  
        type string {  
          length "1 .. max";  
        }  
      }  
  
      leaf mtu {  
        description  
          "The maximum transmission unit (MTU) value assigned to  
          this interface.";  
        type uint32;  
        default 1500;  
      }  
  
      leaf status {  
        description  
          "The current status of this interface.";  
        type status-type;  
      }  
    }  
  }  
}
```

```
        config false;
      }
    }
  }
```

A.2. Example Data Set

The following data element shows the conceptual contents of the example server for the protocol operation examples in the next section. This includes all the configuration data nodes, non-configuration data nodes, and default leafs.

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces xmlns="http://example.com/ns/interfaces">
    <interface>
      <name>eth0</name>
      <mtu>8192</mtu>
      <status>up</status>
    </interface>
    <interface>
      <name>eth1</name>
      <mtu>1500</mtu>
      <status>up</status>
    </interface>
    <interface>
      <name>eth2</name>
      <mtu>9000</mtu>
      <status>not feeling so good</status>
    </interface>
    <interface>
      <name>eth3</name>
      <mtu>1500</mtu>
      <status>waking up</status>
    </interface>
  </interfaces>
</data>
```

In this example, the 'mtu' field for each interface entry is set in the following manner:

| name | set by | mtu |
|------|--------|------|
| eth0 | client | 8192 |
| eth1 | server | 1500 |
| eth2 | client | 9000 |
| eth3 | client | 1500 |

A.3. Protocol Operation Examples

The following examples shows some <get> operations using the 'with-defaults' element. The data model used for these examples is defined in Appendix A.1.

The client is retrieving all the data nodes within the 'interfaces' object, filtered with the <with-defaults> parameter.

A.3.1. <with-defaults> = 'report-all'

The behavior of the <with-defaults> parameter handling for the value 'report-all' is demonstrated in this example.

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-defaults
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">
      report-all
    </with-defaults>
  </get>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <name>eth0</name>
        <mtu>8192</mtu>
        <status>up</status>
      </interface>
      <interface>
        <name>eth1</name>
        <mtu>1500</mtu>
        <status>up</status>
      </interface>
      <interface>
        <name>eth2</name>
        <mtu>9000</mtu>
        <status>not feeling so good</status>
      </interface>
      <interface>
        <name>eth3</name>
        <mtu>1500</mtu>
        <status>waking up</status>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.3.2. <with-defaults> = 'report-all-tagged'

The behavior of the <with-defaults> parameter handling for the value 'report-all-tagged' is demonstrated in this example. A 'tagged' data node is an element that contains the 'default' XML attribute, set to

'true' or '1'.

The actual data nodes tagged by the server depends on the defaults handling basic mode used by the server. Only the data nodes that are considered to be default data will be tagged.

In this example, the server's basic mode is equal to 'trim', so all data nodes that would contain the schema default value are tagged. If the server's basic mode is 'explicit', then only data nodes that are not explicitly set data are tagged. If the server's basic mode is 'report-all', then no data nodes are tagged.

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-defaults
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">
      report-all-tagged
    </with-defaults>
  </get>
</rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:wd="urn:ietf:params:xml:ns:netconf:default:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <name>eth0</name>
        <mtu>8192</mtu>
        <status wd:default="true">up</status>
      </interface>
      <interface>
        <name>eth1</name>
        <mtu wd:default="true">1500</mtu>
        <status wd:default="true">up</status>
      </interface>
      <interface>
        <name>eth2</name>
        <mtu>9000</mtu>
        <status>not feeling so good</status>
      </interface>
      <interface>
        <name>eth3</name>
        <mtu wd:default="true">1500</mtu>
        <status>waking up</status>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.3.3. <with-defaults> = 'trim'

The behavior of the <with-defaults> parameter handling for the value 'trim' is demonstrated in this example.

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-defaults
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">
      trim
    </with-defaults>
  </get>
</rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <name>eth0</name>
        <mtu>8192</mtu>
      </interface>
      <interface>
        <name>eth1</name>
      </interface>
      <interface>
        <name>eth2</name>
        <mtu>9000</mtu>
        <status>not feeling so good</status>
      </interface>
      <interface>
        <name>eth3</name>
        <status>waking up</status>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.3.4. <with-defaults> = 'explicit'

The behavior of the <with-defaults> parameter handling for the value 'explicit' is demonstrated in this example.

```
<rpc message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-defaults
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">
      explicit
    </with-defaults>
  </get>
</rpc>

<rpc-reply message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <name>eth0</name>
        <mtu>8192</mtu>
        <status>up</status>
      </interface>
      <interface>
        <name>eth1</name>
        <status>up</status>
      </interface>
      <interface>
        <name>eth2</name>
        <mtu>9000</mtu>
        <status>not feeling so good</status>
      </interface>
      <interface>
        <name>eth3</name>
        <mtu>1500</mtu>
        <status>waking up</status>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```


Appendix B. Change Log

-- RFC Ed.: remove this section before publication.

B.1. 13-14

Removed reference to RFC 4741 and using 4741bis instead.

B.2. 12-13

Removed with-defaults capability conformance section.

Changed 'wd:default' to 'default'.

Added normative reference to XSD.

Clarified conditional support for with-defaults enumerations, based on capability parameters.

Clarified that all xs:boolean encoding values must be supported.

Clarified purpose of also-supported parameter in capability URI.

B.3. 11-12

Made editorial clarifications based on AD review.

B.4. 10-11

Changed term 'database' to 'configuration datastore' or generic 'datastore'.

B.5. 09-10

Changed term 'datastore' to 'database'.

Added term 'default value'.

Clarified verbage for data node containing a default value.

B.6. 08-09

Removed non-volatile server requirements.

Moved some text from basic-mode section into the the retrieval modes section.

Added description and reference statements to the YANG module.

Many bugfixes and clarifications, based on WGLC review comments.

B.7. 07-08

Added report-all-tagged mode.

Changed conformance so report-all or report-all-tagged mode SHOULD be supported.

Clarified capability requirements for each mode, for edit-config and NV storage requirements.

Changed rpc-error details for unsupported with-defaults value.

Added XSD for wd:default attribute

Expanded example to show report-all-tagged for a basic-mode=trim server.

B.8. 06-07

Removed text in capability identifier section about adding YANG module capability URI parameters.

Changed YANG module namespace to match YANG format, and updated examples to use this new namespace.

Fixed some typos.

B.9. 05-06

Removed ':1.0' from capability URI.

Removed open issues section because all known issues are closed.

Moved examples to a separate appendix, and expanded them.

Added example.yang as an appendix to properly explain the examples used within the document.

Replaced normative term 'SHALL' with 'MUST' to be consistent within this document.

Clarified <with-defaults> behavior for non-configuration data nodes.

Clarified various sections based on WGLC comments on mailing list.

Removed some unused terms.

Reversed the order of the change log sections so the most recent changes are shown first.

B.10. 04-05

Updated I-D and YANG module boiler-plate.

Removed redundant 'with-defaults' YANG feature.

Changed definition of 'explicit' mode to match the YANG definition

Removed XSD because the YANG is normative and the XSD is unconstrained, and does not properly extend the 3 affected NETCONF operations.

Made the YANG module a normative section instead of non-normative appendix.

Changed YANG from an informative to a normative reference,

Changed 4741bis from an informative to a normative reference because the YANG module imports the ietf-netconf module in order to augment some operations.

Updated capability requirements to include YANG module capability parameters.

Added a description statement to the with-defaults leaf definition.

Update open issues section; ready to close all open issues.

B.11. 03-04

Clarifications

Added non-netconf interfaces to the definition of explicitly set default data

B.12. 02-03

Clarifications

YAM added

Use the same URN for the capability and the XML namespace to accommodate YANG, and avoid two separate URN/URIs being advertised in the HELLO message, for such a small function.

B.13. 01-02

report-all made mandatory

Placeholder for YAM added, XSD will be removed when 4741 provides the NETCONF YAM

with-defaults is valid for state data as well (if state data has a defined default which might not be so frequent). The definition of explicit was modified for state data.

B.14. 00-01

Changed value set of with-default capability and element

Added version to URI

B.15. -00

Created from draft-bierman-netconf-with-defaults-01.txt

It was decided by the NETCONF mailing list, that with-defaults should be a sub-element of each affected operation. While this violates the XSD of RFC4741 this is acceptable and follows the ideas behind NETCONF and YANG.

Hopefully it will be clarified in the 4741bis RFC whether such extensions are allowed.

Authors' Addresses

Andy Bierman
Brocade

Email: andy.bierman@brocade.com

Balazs Lengyel
Ericsson
Budapest,
Hungary

Email: balazs.lengyel@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 27, 2011

P. Shafer
Juniper Networks
September 23, 2010

An Architecture for Network Management using NETCONF and YANG
draft-ietf-netmod-arch-10

Abstract

The Network Configuration Protocol (NETCONF) gives access to native capabilities of the devices within a network, defining methods for manipulating configuration databases, retrieving operational data, and invoking specific operations. YANG provides the means to define the content carried via NETCONF, both data and operations. Using both technologies, standard modules can be defined to give interoperability and commonality to devices, while still allowing devices to express their unique capabilities.

This document describes how NETCONF and YANG help build network management applications that meet the needs of network operators.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 27, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

| | |
|---|----|
| 1. Origins of NETCONF and YANG | 4 |
| 2. Elements of the Architecture | 6 |
| 2.1. NETCONF | 6 |
| 2.1.1. NETCONF Transport Mappings | 8 |
| 2.2. YANG | 9 |
| 2.2.1. Constraints | 11 |
| 2.2.2. Flexibility | 12 |
| 2.2.3. Extensibility Model | 12 |
| 2.3. YANG Translations | 13 |
| 2.3.1. YIN | 14 |
| 2.3.2. DSDL (RELAX NG) | 14 |
| 2.4. YANG Types | 15 |
| 2.5. IETF Guidelines | 15 |
| 3. Working with YANG | 16 |
| 3.1. Building NETCONF- and YANG-based Solutions | 16 |
| 3.2. Addressing Operator Requirements | 17 |
| 3.3. Roles in Building Solutions | 20 |
| 3.3.1. Modeler | 20 |
| 3.3.2. Reviewer | 20 |
| 3.3.3. Device Developer | 20 |
| 3.3.4. Application Developer | 21 |
| 4. Modeling Considerations | 24 |
| 4.1. Default Values | 24 |
| 4.2. Compliance | 25 |
| 4.3. Data Distinctions | 26 |
| 4.3.1. Background | 26 |
| 4.3.2. Definitions | 27 |
| 4.3.3. Implications | 28 |
| 4.4. Direction | 29 |
| 5. Security Considerations | 30 |
| 6. IANA Considerations | 31 |
| 7. Normative References | 32 |
| Author's Address | 34 |

1. Origins of NETCONF and YANG

Networks are increasing in complexity and capacity, as well as the density of the services deployed upon them. Uptime, reliability, and predictable latency requirements drive the need for automation. The problems with network management are not simple. They are complex and intricate. But these problems must be solved for networks to meet the stability needs of existing services while incorporating new services in a world where the growth of networks is exhausting the supply of qualified networking engineers.

In June of 2002, Internet Architecture Board (IAB) held a workshop on Network Management ([RFC3535]). The members of this workshop made a number of observations and recommendations for the IETF's consideration concerning the issues operators were facing in their network management-related work as well as issues they were having with the direction of the IETF activities in this area.

The output of this workshop was focused on current problems. The observations were reasonable and straight forward, including the need for transactions, rollback, low implementation costs, and the ability to save and restore the device's configuration data. Many of the observations give insight into the problems operators were having with existing network management solutions, such as the lack of full coverage of device capabilities and the ability to distinguish between configuration data and other types of data.

Based on these directions, the NETCONF working group was formed and the Network Configuration (NETCONF) protocol was created. This protocol defines a simple mechanism where network management applications, acting as clients, can invoke operations on the devices, which act as servers. The NETCONF specification ([RFC4741]) defines a small set of operations, but goes out of its way to avoid making any requirements on the data carried in those operations, preferring to allow the protocol to carry any data. This "data model agnostic" approach allows data models to be defined independently.

But lacking a means of defining data models, the NETCONF protocol was not usable for standards-based work. Existing data modeling languages such as the XML Schema Language (XSD) ([W3CXSD]) and the Document Schema Definition Languages (DSDL) ([ISODSDL]) were considered, but were rejected because the problem domains have little natural overlap. Defining a data model or protocol that is encoded in XML is a distinct problem from defining an XML document. The use of NETCONF operations place requirements on the data content that are not shared with the static document problem domain addressed by schema languages like XSD or RELAX NG.

In 2007 and 2008, the issue of a data modeling language for NETCONF was discussed in the OPS and APPS areas of IETF 70 and 71, and a design team was tasked with creating a requirements document (expired I-D draft-presuhn-rcdml-03.txt). After discussing the available options at the CANMOD BoF at IETF71, the community wrote a charter for the NETMOD working group. An excellent description of this time period is available at <http://www.ietf.org/mail-archive/web/ietf/current/msg51644.html>

In 2008 and 2009, the NETMOD working group produced a specification for YANG ([RFCYANG]) as a means for defining data models for NETCONF, allowing both standard and proprietary data models to be published in a form that is easily digestible by human readers and satisfies many of the issues raised in the IAB NM workshop. This brings NETCONF to a point where it can be used to develop standard data models within the IETF.

YANG allows a modeler to create a data model, to define the organization of the data in that model, and to define constraints on that data. Once published, the YANG module acts as a contract between the client and server, with both parties understanding how their peer will expect them to behave. A client knows how to create valid data for the server, and knows what data will be sent from the server. A server knows the rules that govern the data and how it should behave.

YANG also incorporates a level of extensibility and flexibility not present in other model languages. New modules can augment the data hierarchies defined in other modules, seamlessly adding data at appropriate places in the existing data organization. YANG also allows new statements to be defined, allowing the language itself to be expanded in a consistent way.

This document presents an architecture for YANG, describing how YANG-related technologies work and how solutions built on them can address the network management problem domain.

2. Elements of the Architecture

2.1. NETCONF

NETCONF defines an XML-based remote procedure call (RPC) mechanism that leverages the simplicity and availability of high-quality XML parsers. XML gives a rich, flexible, hierarchical, standard representation of data that matches the needs of networking devices. NETCONF carries configuration data and operations as requests and replies using RPCs encoded in XML over a connection-oriented transport.

XML's hierarchical data representation allows complex networking data to be rendered in a natural way. For example, the following configuration places interfaces in OSPF areas. The <ospf> element contains a list of <area> elements, each of which contain a list of <interface> elements. The <name> element identifies the specific area or interface. Additional configuration for each area or interface appears directly inside the appropriate element.

```
<ospf xmlns="http://example.org/netconf/ospf">

  <area>
    <name>0.0.0.0</name>

    <interface>
      <name>ge-0/0/0.0</name>
      <!-- The priority for this interface -->
      <priority>30</priority>
      <metric>100</metric>
      <dead-interval>120</dead-interval>
    </interface>

    <interface>
      <name>ge-0/0/1.0</name>
      <metric>140</metric>
    </interface>
  </area>

  <area>
    <name>10.1.2.0</name>

    <interface>
      <name>ge-0/0/2.0</name>
      <metric>100</metric>
    </interface>

    <interface>
      <name>ge-0/0/3.0</name>
      <metric>140</metric>
      <dead-interval>120</dead-interval>
    </interface>
  </area>
</ospf>
```

NETCONF includes mechanisms for controlling configuration datastores. Each datastore is a specific collection of configuration data that can be used as source or target of the configuration-related operations. The device can indicate whether it has a distinct "startup" configuration datastore, whether the current or "running" datastore is directly writable, or whether there is a "candidate" configuration datastore where configuration changes can be made that will not affect the device until a "commit-configuration" operation is invoked.

NETCONF defines operations that are invoked as RPCs from the client (the application) to the server (running on the device). The following table lists some of these operations:

| Operation | Description |
|---------------|--|
| commit | Commits the "candidate" configuration to "running" |
| copy-config | Copy one configuration datastore to another |
| delete-config | Delete a configuration datastore |
| edit-config | Change the contents of a configuration datastore |
| get-config | Retrieve all or part of a configuration datastore |
| lock | Prevent changes to a datastore from another party |
| unlock | Release a lock on a datastore |

NETCONF's "capability" mechanism allows the device to announce the set of capabilities that the device supports, including protocol operations, datastores, data models, and other abilities. These are announced during session establishment as part of the <hello> message. A client can inspect the hello message to determine what the device is capable of and how to interact with the device to perform the desired tasks.

NETCONF also defines a means of sending asynchronous notifications from the server to the client, described in [RFC5277].

In addition, NETCONF can fetch state data, receive notifications, and invoke additional RPC methods defined as part of a capability. Complete information about NETCONF can be found in [RFC4741].

2.1.1. NETCONF Transport Mappings

NETCONF can run over any transport protocol that meets the requirements defined in RFC4741, including

- o connection-oriented operation
- o authentication
- o integrity
- o confidentiality

[RFC4742] defines an mapping for the SSH ([RFC4251]) protocol, which is the mandatory transport protocol. Others include SOAP ([RFC4743]), BEEP ([RFC4744]), and TLS ([RFC5539]).

2.2. YANG

YANG is a data modeling language for NETCONF. It allows the description of hierarchies of data nodes ("nodes") and the constraints that exist among them. YANG defines data models and how to manipulate those models via NETCONF protocol operations.

Each YANG module defines a data model, uniquely identified by a namespace URI. These data models are extensible in a manner that allows tight integration of standard data models and proprietary data models. Models are built from organizational containers, lists of data nodes and data node forming leafs of the data tree.

```

module example-ospf {
  namespace "http://example.org/netconf/ospf";
  prefix ospf;

  import network-types { // Access another module's def'ns
    prefix nett;
  }

  container ospf { // Declare the top-level tag
    list area { // Declare a list of "area" nodes
      key name; // The key "name" identifies list members
      leaf name {
        type nett:area-id;
      }
      list interface {
        key name;
        leaf name {
          type nett:interface-name;
        }
        leaf priority {
          description "Designated router priority";
          type uint8; // The type is a constraint on
                      // valid values for "priority".
        }
        leaf metric {
          type uint16 {
            range 1..65535;
          }
        }
        leaf dead-interval {
          units seconds;
          type uint16 {
            range 1..65535;
          }
        }
      }
    }
  }
}

```

A YANG module defines a data model in terms of the data, its hierarchical organization, and the constraints on that data. YANG defines how this data is represented in XML and how that data is used in NETCONF operations.

The following table briefly describes some common YANG statements:

| Statement | Description |
|--------------|--|
| augment | Extends existing data hierarchies |
| choice | Defines mutually exclusive alternatives |
| container | Defines a layer of the data hierarchy |
| extension | Allows new statements to be added to YANG |
| feature | Indicates parts of the model are optional |
| grouping | Groups data definitions into reusable sets |
| key | Defines the key leafs for lists |
| leaf | Defines a leaf node in the data hierarchy |
| leaf-list | A leaf node that can appear multiple times |
| list | A hierarchy that can appear multiple times |
| notification | Defines notification |
| rpc | Defines input and output parameters for an RPC operation |
| typedef | Defines a new type |
| uses | Incorporates the contents of a "grouping" |

2.2.1. Constraints

YANG allows the modeler to add constraints to the data model to prevent impossible or illogical data. These constraints give clients information about the data being sent from the device, and also allow the client to know as much as possible about the data the device will accept, so the client can send correct data. These constraints apply to configuration data, but can also be used for rpc and notification data.

The principal constraint is the "type" statement, which limits the contents of a leaf node to that of the named type. The following table briefly describes some other common YANG constraints:

| Statement | Description |
|--------------|--|
| length | Limits the length of a string |
| mandatory | Requires the node appear |
| max-elements | Limits the number of instances in a list |
| min-elements | Limits the number of instances in a list |
| must | XPath expression must be true |
| pattern | Regular expression must be satisfied |
| range | Value must appear in range |
| reference | Value must appear elsewhere in the data |
| unique | Value must be unique within the data |
| when | Node is only present when XPath expression is true |

The "must" and "when" statements use XPath ([W3CXPATH]) expressions to specify conditions that are semantically evaluated against the data hierarchy, but neither the client nor the server are required to implement the XPath specification. Instead they can use any means to ensure these conditions are met.

2.2.2. Flexibility

YANG uses the "union" type and the "choice" and "feature" statements to give modelers flexibility in defining their data models. The "union" type allows a single leaf to accept multiple types, like an integer or the word "unbounded":

```
type union {  
    type int32;  
    type enumeration {  
        enum "unbounded";  
    }  
}
```

The "choice" statement lists a set of mutually exclusive nodes, so a valid configuration can choose any one node (or case). The "feature" statement allows the modeler to identify parts of the model which can be optional, and allows the device to indicate whether it implements these optional portions.

The "deviation" statement allows the device, to indicate parts of a YANG module which the device does not faithfully implement. While devices are encouraged to fully abide according to the contract presented in the YANG module, real world situations may force the device to break the contract. Deviations give a means of declaring this limitation, rather than leaving it to be discovered via run-time errors.

2.2.3. Extensibility Model

XML includes the concept of namespaces, allowing XML elements from different sources to be combined in the same hierarchy without risking collision. YANG modules define content for specific namespaces, but one module may augment the definition of another module, introducing elements from that module's namespace into the first module's hierarchy.

Since one module can augment another module's definition, hierarchies of definitions are allowed to grow, as definitions from multiple sources are added to the base hierarchy. These augmentations are qualified using the namespace of the source module, helping to avoid issues with name conflicts as the modules change over time.

For example, if the above OSPF configuration were the standard, a vendor module may augment this with vendor-specific extensions.

```
module vendorx-ospf {
  namespace "http://vendorx.example.com/ospf";
  prefix vendorx;

  import example-ospf {
    prefix ospf;
  }

  augment /ospf:ospf/ospf:area/ospf:interfaces {
    leaf no-neighbor-down-notification {
      type empty;
      description "Don't inform other protocols about "
        + " neighbor down events";
    }
  }
}
```

The <no-neighbor-down-notification> element is then placed in the vendorx namespace:

```
<ospf xmlns="http://example.org/netconf/ospf"
      xmlns:vendorx="http://vendorx.example.com/ospf">

  <area>
    <name>0.0.0.0</name>

    <interface>
      <name>ge-0/0/0.0</name>
      <priority>30</priority>
      <vendorx:no-neighbor-down-notification/>
    </interface>

  </area>
</ospf>
```

Augmentations are seamlessly integrated with base modules, allowing them to be fetched, archived, loaded, and deleted within their natural hierarchy. If a client application asks for the configuration for a specific OSPF area, it will receive the sub-hierarchy for that area, complete with any augmented data.

2.3. YANG Translations

The YANG data modeling language is the central piece of a group of related technologies. The YANG language itself, described in

[RFCYANG], defines the syntax of the language and its statements, the meaning of those statements, and how to combine them to build the hierarchy of nodes that describe a data model.

That document also defines the "on the wire" XML content for NETCONF operations on data models defined in YANG modules. This includes the basic mapping between YANG data tree nodes and XML elements, as well as mechanisms used in <edit-config> content to manipulate that data, such as arranging the order of nodes within a list.

YANG uses a syntax that is regular and easily described, primarily designed for human readability. YANG's syntax is friendly to email, diff, patch, and the constraints of RFC formatting.

2.3.1. YIN

In some environments, incorporating a YANG parser may not be an acceptable option. For those scenarios, an XML grammar for YANG is defined as YIN (YANG Independent Notation). YIN allows the use of XML parsers which are readily available in both open source and commercial versions. Conversion between YANG and YIN is direct, loss-less and reversible. YANG statements are converted to XML elements, preserving the structure and content of YANG, but enabling the use of off-the-shelf XML parsers rather than requiring the integration of a YANG parser. YIN maintains complete semantic equivalence with YANG.

2.3.2. DSDL (RELAX NG)

Since NETCONF content is encoded in XML, it is natural to use XML schema languages for their validation. To facilitate this, YANG offers a standardized mapping of YANG modules into Document Schema Description Languages ([RFCYANGDSDL]), of which RELAX NG is a major component.

DSDL is considered to be the best choice as a standard schema language because it addresses not only grammar and datatypes of XML documents but also semantic constraints and rules for modifying the information set of the document.

In addition, DSDL offers formal means for coordinating multiple independent schemas and specifying how to apply the schemas to the various parts of the document. This is useful since YANG content is typically composed of multiple vocabularies.

2.4. YANG Types

YANG supports a number of builtin types, and allows additional types to be derived from those types in an extensible manner. New types can add additional restrictions to allowable data values.

A standard type library for use by YANG is available [RFCYANGTYPES]. These YANG modules define commonly used data types for IETF-related standards.

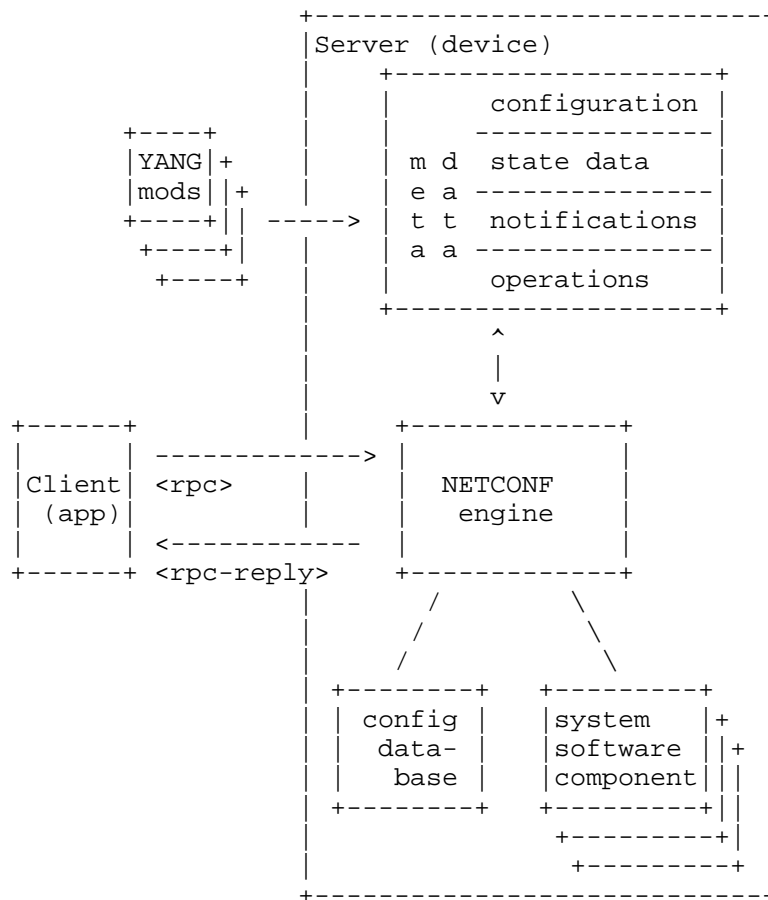
2.5. IETF Guidelines

A set of additional guidelines are defined that indicate desirable usage for authors and reviewers of standards track specifications containing YANG data model modules ([RFCYANGUSAGE]). These guidelines should be used as a basis for reviews of other YANG data model documents.

3. Working with YANG

3.1. Building NETCONF- and YANG-based Solutions

In the typical YANG-based solution, the client and server are driven by the content of YANG modules. The server includes the definitions of the modules as meta-data that is available to the NETCONF engine. This engine processes incoming requests, uses the meta-data to parse and verify the request, performs the requested operation, and returns the results to the client.



To use YANG, YANG modules must be defined to model the specific problem domain. These modules are then loaded, compiled, or coded into the server.

The sequence of events for the typical client/server interaction may

be as follows:

- o A client application ([C]) opens a NETCONF session to the server (device) ([S])
- o [C] and [S] exchange <hello> messages containing the list of capabilities supported by each side, allowing [C] to learn the modules supported by [S]
- o [C] builds and sends an operation defined in the YANG module, encoded in XML, within NETCONF's <rpc> element
- o [S] receives and parses the <rpc> element
- o [S] verifies the contents of the request against the data model defined in the YANG module
- o [S] performs the requested operation, possibly changing the configuration datastore
- o [S] builds the response, containing the response, any requested data, and any errors
- o [S] sends the response, encoded in XML, within NETCONF's <rpc-reply> element
- o [C] receives and parses the <rpc-reply> element
- o [C] inspects the response and processes it as needed

Note that there is no requirement for the client or server to process the YANG modules in this way. The server may hard code the contents of the data model, rather than handle the content via a generic engine. Or the client may be targeted at the specific YANG model, rather than being driven generically. Such a client might be a simple shell script that stuffs arguments into an XML payload template and sends it to the server.

3.2. Addressing Operator Requirements

NETCONF and YANG address many of the issues raised in the IAB NM workshop.

- o Ease of use: YANG is designed to be human friendly, simple and readable. Many tricky issues remain due to the complexity of the problem domain, but YANG strives to make them more visible and easier to deal with.

- o Configuration and state data: YANG clearly divides configuration data from other types of data.
- o Transactions: NETCONF provides a simple transaction mechanism.
- o Generation of deltas: A YANG module gives enough information to generate the delta needed to change between two configuration data sets.
- o Dump and restore: NETCONF gives the ability to save and restore configuration data. This can also be performed for a specific YANG module.
- o Network-wide configuration: NETCONF supports robust network-wide configuration transactions via the commit and confirmed-commit capability. When a change is attempted that affects multiple devices, these capabilities simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.
- o Text-friendly: YANG modules are very text friendly, as is the data they define.
- o Configuration handling: NETCONF addresses the ability to distinguish between distributing configuration data and activating it.
- o Task-oriented: A YANG module can define specific tasks as RPC operations. A client can choose to invoke the RPC operation or to access any underlying data directly.
- o Full coverage: YANG modules can be defined that give full coverage to all the native abilities of the device. Giving this access avoids the need to resort to the command line interface (CLI) using tools such as Expect ([SWEXPECT]).
- o Timeliness: YANG modules can be tied to CLI operations, so all native operations and data are immediately available.
- o Implementation difficulty: YANG's flexibility enables modules that can be more easily implemented. Adding "features" and replacing "third normal form" with a natural data hierarchy should reduce complexity.
- o Simple data modeling language: YANG has sufficient power to be usable in other situations. In particular, on-box API and native CLI can be integrated to achieve simplification of the infrastructure.

- o Internationalization: YANG uses UTF-8 ([RFC3629]) encoded unicode characters.
- o Event correlation: YANG integrates RPC operations, notification, configuration and state data, enabling internal references. For example, a field in a notification can be tagged as pointing to a BGP peer, and the client application can easily find that peer in the configuration data.
- o Implementation costs: Significant effort has been made to keep implementation costs as low as possible.
- o Human friendly syntax: YANG's syntax is optimized for the reader, specifically the reviewer on the basis that this is the most common human interaction.
- o Post-processing: Use of XML will maximize the opportunities for post-processing of data, possibly using XML-based technologies like XPath ([W3CXPATH], XQuery ([W3CXQUERY]), and XSLT ([W3CXSLT]).
- o Semantic mismatch: Richer, more descriptive data models will reduce the possibility of semantic mismatch. With the ability to define new primitives, YANG modules will be more specific in content, allowing more enforcement of rules and constraints.
- o Security: NETCONF runs over transport protocols secured by SSH or TLS, allowing secure communications and authentication using well-trusted technology. The secure transport can use existing key and credential management infrastructure, reducing deployment costs.
- o Reliable: NETCONF and YANG are solid and reliable technologies. NETCONF is connection based, and includes automatic recovery mechanisms when the connection is lost.
- o Delta friendly: YANG-based models support operations that are delta friendly. Add, change, insert, and delete operations are all well defined.
- o Method-oriented: YANG allows new RPC operations to be defined, including an operation name, which is essentially a method. The input and output parameters of the RPC operations are also defined in the YANG module.

3.3. Roles in Building Solutions

Building NETCONF- and YANG-based solutions requires interacting with many distinct groups. Modelers must understand how to build useful models that give structure and meaning to data while maximizing the flexibility of that data to "future proof" their work. Reviewers need to quickly determine if that structure is accurate. Device developers need to code that data model into their devices, and application developers need to code their applications to take advantage of that data model. There are a variety of strategies for performing each piece of this work. This section discusses some of those strategies.

3.3.1. Modeler

The modeler defines a data model based on their in-depth knowledge of the problem domain being modeled. This model should be as simple as possible, but should balance complexity with expressiveness. The organization of the model should target not only the current model, but should allow for extensibility from other modules and for adaptability to future changes.

Additional modeling issues are discussed in Section 4.

3.3.2. Reviewer

The reviewer role is perhaps the most important and the time reviewers are willing to give is precious. To help the reviewer, YANG stresses readability, with a human-friendly syntax, natural data hierarchy, and simple, concise statements.

3.3.3. Device Developer

The YANG model tells the device developer what data is being modeled. The developer reads the YANG models and writes code that supports the model. The model describes the data hierarchy and associated constraints, and the description and reference material helps the developer understand how to transform the models view into the device's native implementation.

3.3.3.1. Generic Content Support

The YANG model can be compiled into a YANG-based engine for either the client or server side. Incoming data can be validated, as can outgoing data. The complete configuration datastore may be validated in accordance with the constraints described in the data model.

Serializers and deserializers for generating and receiving NETCONF

content can be driven by the meta-data in the model. As data is received, the meta-data is consulted to ensure the validity of incoming XML elements.

3.3.3.2. XML Definitions

The YANG module dictates the XML encoding for data sent via NETCONF. The rules that define the encoding are fixed, so the YANG module can be used to ascertain whether a specific NETCONF payload is obeying the rules.

3.3.4. Application Developer

The YANG module tells the application developer what data can be modeled. Developers can inspect the modules and take one of three distinct views. In this section, we will consider them and the impact of YANG on their design. In the real world, most applications are a mixture of these approaches.

3.3.4.1. Hard Coded

An application can be coded against the specific, well-known contents of YANG modules, implementing their organization, rules, and logic directly with explicit knowledge. For example, a script could be written to change the domain name of a set of devices using a standard YANG module that includes such a leaf node. This script takes the new domain name as an argument and inserts it into a string containing the rest of the XML encoding as required by the YANG module. This content is then sent via NETCONF to each of the devices.

This type of application is useful for small, fixed problems where the cost and complexity of flexibility is overwhelmed by the ease of hard coding direct knowledge into the application.

3.3.4.2. Bottom Up

An application may take a generic, bottom up approach to configuration, concentrating on the device's data directly and treating that data without specific understanding.

YANG modules may be used to drive the operation of the YANG equivalent of a "MIB Browser". Such an application manipulates the device's configuration data based on the data organization contained in the YANG module. For example, a GUI may present a straight-forward visualization where elements of the YANG hierarchy are depicted in a hierarchy of folders or GUI panels. Clicking on a line expands to the contents of the matching XML hierarchy.

This type of GUI can easily be built by generating XSLT stylesheets from the YANG data models. An XSLT engine can then be used to turn configuration data into a set of web pages.

The YANG modules allow the application to enforce a set of constraints without understanding the semantics of the YANG module.

3.3.4.3. Top Down

In contrast to the bottom-up approach, the top-down approach allows the application to take a view of the configuration data which is distinct from the standard and/or proprietary YANG modules. The application is free to construct its own model for data organization and to present this model to the user. When the application needs to transmit data to a device, the application transforms its data from the problem-oriented view of the world into the data needed for that particular device. This transformation is under the control and maintenance of the application, allowing the transformation to be changed and updated without affecting the device.

For example, an application could be written that models VPNs in a network-oriented view. The application would need to transform these high-level VPN definitions into the configuration data that would be handed to any particular device within a VPN.

Even in this approach, YANG is useful since it can be used to model the VPN. For example, the following VPN straw-man models a list of VPNs, each with a protocol, a topology, a list of member interfaces, and a list of classifiers.

```
list example-bgpvpn {
  key name;
  leaf name { ... }
  leaf protocol {
    type enumeration {
      enum bgpvpn;
      enum l2vpn;
    }
  }
  leaf topology {
    type enumeration {
      enum hub-n-spoke;
      enum mesh;
    }
  }
  list members {
    key "device interface";
    leaf device { ... }
    leaf interface { ... }
  }
  list classifiers {
    ...
  }
}
```

The application can use such a YANG module to drive its operation, building VPN instances in a database and then pushing the configuration for those VPNs to individual devices using either a standard device model (e.g. example-bgpvpn.yang) or by transforming that standard device content into some proprietary format for devices that do not support that standard.

4. Modeling Considerations

This section discusses considerations the modeler should be aware of while developing models in YANG.

4.1. Default Values

The concept of default values is simple, but their details, representation, and interaction with configuration data can be difficult issues. NETCONF leaves default values as a data model issue, and YANG gives flexibility to the device implementation in terms of how default values are handled. The requirement is that the device "MUST operationally behave as if the leaf was present in the data tree with the default value as its value". This gives the device implementation choices in how default values are handled.

One choice is to view the configuration as a set of instructions for how the device should be configured. If a data value that is given as part of those instructions is the default value, then it should be retained as part of the configuration, but if it is not explicitly given, then the value is not considered to be part of configuration.

Another choice is to trim values that are identical to the default values, implicitly removing them from the configuration datastore. The act of setting a leaf to its default value effectively deletes that leaf.

The device could also choose to report all default values, regardless of whether they were explicitly set. This choice eases the work of a client that needs default values, but may significantly increase the size of the configuration data.

These choices reflect the default handling schemes of widely deployed networking devices and supporting them allows YANG to reduce implementation and deployment costs of YANG-based models.

When the client retrieves data from the device, it must be prepared to handle the absence of leaf nodes with the default value, since the server is not required to send such leaf elements. This permits the device to implement either of the first two default handling schemes given above.

Regardless of the implementation choice, the device can support the "with-defaults" capability ([RFCWITHDEFAULTS]) and give the client the ability to select the desired handling of default values.

When evaluating the XPath expressions for constraints like "must" and "when", the evaluation context for the expressions will include any

appropriate default values, so the modeler can depend on consistent behavior from all devices.

4.2. Compliance

In developing good data models, there are many conflicting interests the data modeler must keep in mind. Modelers need to be aware of five issues with models and devices:

- o usefulness
- o compliance
- o flexibility
- o extensibility
- o deviations

For a model to be interesting, it must be useful, solving a problem in a more direct or more powerful way than can be accomplished without the model. The model should maximize the usefulness of the model within the problem domain.

Modelers should build models that maximize the number of devices that can faithfully implement the model. If the model is drawn too narrowly, or includes too many assumptions about the device, then the difficulty and cost of accurately implementing the model will lead to low quality implementations, interoperability issues, and will reduce the value of the model.

Modelers can use the "feature" statement in their models to give the device some flexibility by partitioning their model and allowing the device to indicate which portions of the model are implemented on the device. For example, if the model includes some a "logging" feature, a device with no storage facilities for the log can tell the client that it does not support this feature of the model.

Models can be extended via the "augment" statement, and the modeler should consider how their model is likely to be extended. These augmentations can be defined by vendors, applications, or standards bodies.

Deviations are a means of allowing the devices to indicate where its implementation is not in full compliance with the model. For example, once a model is published, an implementer may decide to make a particular node configurable, where the standard model describes it as state data. The implementation reports the value normally and may

declare a deviation that this device behaves in a different manner than the standard. Applications capable of discovering this deviation can make allowances, but applications that do not discover the deviation can continue treating the implementation as if it were compliant.

Rarely, implementations may make decisions that prevent compliance with the standard. Such occasions are regrettable, but they remain a part of reality, and modelers and application writers ignore them at their own risk. An implementation that emits an integer leaf as "cow" would be difficult to manage, but applications should expect to encounter such misbehaving devices in the field.

Despite this, both client and server should view the YANG module as a contract, with both sides agreeing to abide by the terms. The modeler should be explicit about the terms of such a contract, and both client and server implementations should strive to faithfully and accurately implement the data model described in the YANG module.

4.3. Data Distinctions

The distinction between configuration data, operational state data, and statistics is important to understand for data model writers and people who plan to extend the NETCONF protocol. This section first discusses some background and then provides a definition and some examples.

4.3.1. Background

During the IAB NM workshop, operators did formulate the following two requirements:

2. It is necessary to make a clear distinction between configuration data, data that describes operational state and statistics. Some devices make it very hard to determine which parameters were administratively configured and which were obtained via other mechanisms such as routing protocols.
3. It is required to be able to fetch separately configuration data, operational state data, and statistics from devices, and to be able to compare these between devices.

The NETCONF protocol defined in RFC 4741 distinguishes two types of data, namely configuration data and state data:

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

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

NETCONF does not follow the distinction formulated by the operators between configuration data, operational state data, and statistical data, since it considers state data to include both statistics and operational state data.

4.3.2. Definitions

Below is a definition for configuration data, operational state data, and statistical data. The definition borrows from previous work.

- o Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state. [RFC4741]
- o Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behaviour similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.
- o Statistical data is the set of read-only data created by a system itself. It describes the performance of the system and its components.

The following examples help to clarify the difference between configuration data, operational state data and statistical data.

4.3.2.1. Example 1: IP Routing Table

IP routing tables can contain entries that are statically configured (configuration data) as well as entries obtained from routing protocols such as OSPF (operational state data). In addition, a routing engine might collect statistics like how often a particular routing table entry has been used.

4.3.2.2. Example 2: Interfaces

Network interfaces usually come with a large number of attributes that are specific to the interface type and in some cases specific to the cable plugged into an interface. Examples are the maximum

transmission unit of an interface or the speed detected by an Ethernet interface.

In many deployments, systems use the interface attributes detected when an interface is initialized. As such, these attributes constitute operational state. However, there are usually provisions to overwrite the discovered attributes with static configuration data, like for example configuring the interface MTU to use a specific value or forcing an Ethernet interface to run at a given speed.

The system will record statistics (counters) measuring the number of packets, bytes, and errors received and transmitted on each interface.

4.3.2.3. Example 3: Account Information

Systems usually maintain static configuration information about the accounts on the system. In addition, systems can obtain information about accounts from other sources (e.g. LDAP, NIS) dynamically, leading to operational state data. Information about account usage are examples of statistic data.

Note that configuration data supplied to a system in order to create a new account might be supplemented with additional configuration information determined by the system when the account is being created (such as a unique account id). Even though the system might create such information, it usually becomes part of the static configuration of the system since this data is not transient.

4.3.3. Implications

The primary focus of YANG is configuration data. There is no single mechanism defined for the separation of operational state data and statistics since NETCONF treats them both as state data. This section describes several different options for addressing this issue.

4.3.3.1. Data Models

The first option is to have data models that explicitly differentiate between configuration data and operational state data. This leads to duplication of data structures and might not scale well from a modeling perspective.

For example, the configured duplex value and the operational duplex value would be distinct leafs in the data model.

4.3.3.2. Additional Operations to Retrieve Operational State

The NETCONF protocol can be extended with new protocol operations that specifically allow the retrieval of all operational state, e.g. by introducing a <get-ops> operation (and perhaps also a <get-stats> operation).

4.3.3.3. Introduction of an Operational State Datastore

Another option could be to introduce a new "configuration" data store that represents the operational state. A <get-config> operation on the <operational> data store would then return the operational state determining the behaviour of the box instead of its static and explicit configuration state.

4.4. Direction

At this time, the only viable solution is to distinctly model the configuration and operational values. The configuration leaf would indicate the desired value, as given by the user, and the operational leaf would indicate the current value, as observed on the device.

In the duplex example, this would result in two distinct leafs being defined, "duplex" and "op-duplex", one with "config true" and one with "config false".

In some cases, distinct leafs would be used, but in others, distinct lists might be used. Distinct lists allows the list to be organized in different ways, with different constraints. Keys, sorting, and constraint statements like must, unique, or when may differ between configuration data and operational data.

For example, configured static routes might be a distinct list from the operational routing table, since the use of keys and sorting might differ.

5. Security Considerations

This document discusses an architecture for network management using NETCONF and YANG. It has no security impact on the Internet.

6. IANA Considerations

This document has no actions for IANA.

7. Normative References

- [ISODSDL] International Organization for Standardization, "Document Schema Definition Languages (DSDL) - Part 1: Overview", ISO/IEC 19757-1, November 2004.
- [RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.
- [RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.
- [RFC4743] Goddard, T., "Using NETCONF over the Simple Object Access Protocol (SOAP)", RFC 4743, December 2006.
- [RFC4744] Lear, E. and K. Crozier, "Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)", RFC 4744, December 2006.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.
- [RFCWITHDEFAULTS] Bierman, A. and B. Lengyel, "With-defaults capability for NETCONF", draft-ietf-netconf-with-defaults-11.txt (work in progress).
- [RFCYANG] Bjorklund, M., Ed., "YANG - A data modeling language for the Network Configuration Protocol (NETCONF)", draft-ietf-netmod-yang-13 (work in progress).
- [RFCYANGDSDL] Lhotka, L., Mahy, R., and S. Chishom, "Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content", draft-ietf-netmod-dsdl-map-07 (work in progress).

progress).

[RFCYANGTYPES]

Schoenwaelder, J., "Common YANG Data Types",
draft-ietf-netmod-yang-types-09.txt (work in progress).

[RFCYANGUSAGE]

Bierman, A., "Guidelines for Authors and Reviewers of YANG
Data Model Documents", draft-ietf-netmod-yang-usage-10.txt
(work in progress).

[SWEXPECT]

"The Expect Home Page", <<http://expect.sourceforge.net/>>.

[W3CXPATH]

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

[W3CXQUERY]

Boag, S., "XQuery 1.0: An XML Query Language", W3C WD WD-
xquery-20050915, September 2005.

[W3CXSD]

Walmsley, P. and D. Fallside, "XML Schema Part 0: Primer
Second Edition", World Wide Web Consortium
Recommendation REC-xmlschema-0-20041028, October 2004,
<<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>>.

[W3CXSLT]

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

Author's Address

Phil Shafer
Juniper Networks

Email: phil@juniper.net

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2011

L. Lhotka, Ed.
CESNET
October 21, 2010

Mapping YANG to Document Schema Definition Languages and Validating
NETCONF Content
draft-ietf-netmod-dsdl-map-10

Abstract

This document specifies the mapping rules for translating YANG data models into Document Schema Definition Languages (DSDL), a coordinated set of XML schema languages standardized as ISO/IEC 19757. The following DSDL schema languages are addressed by the mapping: RELAX NG, Schematron and DSRL. The mapping takes one or more YANG modules and produces a set of DSDL schemas for a selected target document type - datastore content, NETCONF message etc. Procedures for schema-based validation of such documents are also discussed.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 6 |
| 2. Terminology and Notation | 8 |
| 2.1. Glossary of New Terms | 11 |
| 3. Objectives and Motivation | 12 |
| 4. DSDL Schema Languages | 14 |
| 4.1. RELAX NG | 14 |
| 4.2. Schematron | 15 |
| 4.3. Document Semantics Renaming Language (DSRL) | 16 |
| 5. Additional Annotations | 17 |
| 5.1. Dublin Core Metadata Elements | 17 |
| 5.2. RELAX NG DTD Compatibility Annotations | 17 |
| 5.3. NETMOD-Specific Annotations | 18 |
| 6. Overview of the Mapping | 20 |
| 7. NETCONF Content Validation | 22 |
| 8. Design Considerations | 23 |
| 8.1. Hybrid Schema | 23 |
| 8.2. Modularity | 25 |
| 8.3. Granularity | 27 |
| 8.4. Handling of XML Namespaces | 27 |
| 9. Mapping YANG Data Models to the Hybrid Schema | 29 |
| 9.1. Occurrence Rules for Data Nodes | 29 |
| 9.1.1. Optional and Mandatory Nodes | 30 |
| 9.1.2. Implicit Nodes | 31 |
| 9.2. Mapping YANG Groupings and Typedefs | 32 |
| 9.2.1. YANG Refinements and Augments | 33 |
| 9.2.2. Type Derivation Chains | 36 |
| 9.3. Translation of XPath Expressions | 38 |
| 9.4. YANG Language Extensions | 39 |
| 10. Mapping YANG Statements to the Hybrid Schema | 41 |
| 10.1. The 'anyxml' Statement | 41 |
| 10.2. The 'argument' Statement | 42 |
| 10.3. The 'augment' Statement | 43 |
| 10.4. The 'base' Statement | 43 |
| 10.5. The 'belongs-to' Statement | 43 |
| 10.6. The 'bit' Statement | 43 |
| 10.7. The 'case' Statement | 43 |
| 10.8. The 'choice' Statement | 43 |
| 10.9. The 'config' Statement | 44 |
| 10.10. The 'contact' Statement | 44 |

| | | |
|----------|---|----|
| 10.11. | The 'container' Statement | 44 |
| 10.12. | The 'default' Statement | 44 |
| 10.13. | The 'description' Statement | 46 |
| 10.14. | The 'deviation' Statement | 46 |
| 10.15. | The 'enum' Statement | 46 |
| 10.16. | The 'error-app-tag' Statement | 46 |
| 10.17. | The 'error-message' Statement | 46 |
| 10.18. | The 'extension' Statement | 46 |
| 10.19. | The 'feature' Statement | 46 |
| 10.20. | The 'grouping' Statement | 46 |
| 10.21. | The 'identity' Statement | 47 |
| 10.22. | The 'if-feature' Statement | 48 |
| 10.23. | The 'import' Statement | 49 |
| 10.24. | The 'include' Statement | 49 |
| 10.25. | The 'input' Statement | 49 |
| 10.26. | The 'key' Statement | 49 |
| 10.27. | The 'leaf' Statement | 49 |
| 10.28. | The 'leaf-list' Statement | 50 |
| 10.29. | The 'length' Statement | 50 |
| 10.30. | The 'list' Statement | 51 |
| 10.31. | The 'mandatory' Statement | 52 |
| 10.32. | The 'max-elements' Statement | 52 |
| 10.33. | The 'min-elements' Statement | 52 |
| 10.34. | The 'module' Statement | 52 |
| 10.35. | The 'must' Statement | 53 |
| 10.36. | The 'namespace' Statement | 53 |
| 10.37. | The 'notification' Statement | 54 |
| 10.38. | The 'ordered-by' Statement | 54 |
| 10.39. | The 'organization' Statement | 54 |
| 10.40. | The 'output' Statement | 54 |
| 10.41. | The 'path' Statement | 54 |
| 10.42. | The 'pattern' Statement | 54 |
| 10.43. | The 'position' Statement | 55 |
| 10.44. | The 'prefix' Statement | 55 |
| 10.45. | The 'presence' Statement | 55 |
| 10.46. | The 'range' Statement | 55 |
| 10.47. | The 'reference' Statement | 55 |
| 10.48. | The 'require-instance' Statement | 55 |
| 10.49. | The 'revision' Statement | 55 |
| 10.50. | The 'rpc' Statement | 55 |
| 10.51. | The 'status' Statement | 56 |
| 10.52. | The 'submodule' Statement | 56 |
| 10.53. | The 'type' Statement | 56 |
| 10.53.1. | The "empty" Type | 57 |
| 10.53.2. | The "boolean" Type | 57 |
| 10.53.3. | The "binary" Type | 58 |
| 10.53.4. | The "bits" Type | 58 |
| 10.53.5. | The "enumeration" and "union" Types | 58 |

| | | |
|-------------|--|----|
| 10.53.6. | The "identityref" Type | 58 |
| 10.53.7. | The "instance-identifier" Type | 59 |
| 10.53.8. | The "leafref" Type | 59 |
| 10.53.9. | The Numeric Types | 59 |
| 10.53.10. | The "string" Type | 61 |
| 10.53.11. | Derived Types | 62 |
| 10.54. | The 'typedef' Statement | 63 |
| 10.55. | The 'unique' Statement | 63 |
| 10.56. | The 'units' Statement | 64 |
| 10.57. | The 'uses' Statement | 64 |
| 10.58. | The 'value' Statement | 64 |
| 10.59. | The 'when' Statement | 64 |
| 10.60. | The 'yang-version' Statement | 64 |
| 10.61. | The 'yin-element' Statement | 64 |
| 11. | Mapping the Hybrid Schema to DSDL | 65 |
| 11.1. | Generating RELAX NG Schemas for Various Document Types | 65 |
| 11.2. | Mapping Semantic Constraints to Schematron | 66 |
| 11.2.1. | Constraints on Mandatory Choice | 69 |
| 11.3. | Mapping Default Values to DSRL | 70 |
| 12. | Mapping NETMOD-specific Annotations to DSDL Schema Languages | 75 |
| 12.1. | The @nma:config Annotation | 75 |
| 12.2. | The @nma:default Annotation | 75 |
| 12.3. | The <nma:error-app-tag> Annotation | 75 |
| 12.4. | The <nma:error-message> Annotation | 75 |
| 12.5. | The @if-feature Annotation | 75 |
| 12.6. | The @nma:implicit Annotation | 76 |
| 12.7. | The <nma:instance-identifier> Annotation | 76 |
| 12.8. | The @nma:key Annotation | 76 |
| 12.9. | The @nma:leaf-list Annotation | 76 |
| 12.10. | The @nma:leafref Annotation | 77 |
| 12.11. | The @nma:min-elements Annotation | 77 |
| 12.12. | The @nma:max-elements Annotation | 77 |
| 12.13. | The <nma:must> Annotation | 77 |
| 12.14. | The <nma:ordered-by> Annotation | 78 |
| 12.15. | The <nma:status> Annotation | 78 |
| 12.16. | The @nma:unique Annotation | 78 |
| 12.17. | The @nma:when Annotation | 78 |
| 13. | IANA Considerations | 79 |
| 14. | Security Considerations | 80 |
| 15. | Contributors | 81 |
| 16. | Acknowledgments | 82 |
| 17. | References | 83 |
| 17.1. | Normative References | 83 |
| 17.2. | Informative References | 84 |
| Appendix A. | RELAX NG Schema for NETMOD-Specific Annotations | 86 |
| Appendix B. | Schema-Independent Library | 91 |
| Appendix C. | Mapping DHCP Data Model - A Complete Example | 92 |

| | | |
|-------------|---|-----|
| C.1. | Input YANG Module | 92 |
| C.2. | Hybrid Schema | 94 |
| C.3. | Final DSDL Schemas | 99 |
| C.3.1. | Main RELAX NG Schema for <nc:get> Reply | 100 |
| C.3.2. | RELAX NG Schema - Global Named Pattern Definitions | 102 |
| C.3.3. | Schematron Schema for <nc:get> Reply | 104 |
| C.3.4. | DSRL Schema for <nc:get> Reply | 106 |
| Appendix D. | Change Log | 107 |
| D.1. | Changes Between Versions -07 and -08 | 107 |
| D.2. | Changes Between Versions -06 and -07 | 107 |
| D.3. | Changes Between Versions -05 and -06 | 107 |
| D.4. | Changes Between Versions -04 and -05 | 108 |
| D.5. | Changes Between Versions -03 and -04 | 108 |
| D.6. | Changes Between Versions -02 and -03 | 109 |
| D.7. | Changes Between Versions -01 and -02 | 110 |
| D.8. | Changes Between Versions -00 and -01 | 110 |
| Author's | Address | 112 |

1. Introduction

The NETCONF Working Group has completed a base protocol used for configuration management [RFC4741]. This base specification defines protocol bindings and an XML container syntax for configuration and management operations, but does not include a data modeling language or accompanying rules for how to model configuration and state information carried by NETCONF. The IETF Operations Area has a long tradition of defining data for SNMP Management Information Bases (MIB) modules [RFC1157] using the Structure of Management Information (SMI) language [RFC2578] 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 without the 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 design a modeling language defining the semantics of operational data, configuration data, event notifications and operations, with focus on "human-friendliness", i.e., readability and ease of use. The result is the YANG data modeling language [RFC6020], which now serves for the normative description of NETCONF data models.

Since NETCONF uses XML for encoding its messages, 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 [DSDL]. 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 - Regular Language for XML Next Generation (RELAX NG), Schematron and Document Schema Renaming Language (DSRL) - already have the status of an ISO/IEC International Standard and are supported in a number of software tools.

This document contains a 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, signatures of remote procedure call (RPC) operations and notifications is expressed as the so-called "hybrid schema" - a single RELAX NG schema with 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 be used for validating specific XML documents such as client requests, server responses or notifications, perhaps also taking into account additional context such as active capabilities or features.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [RFC4741]:

- o client
- o datastore
- o message
- o operation
- o server

The following terms are defined in [RFC6020]:

- o augment
- o base type
- o built-in type
- o configuration data
- o container
- o data model
- o data node
- o data tree
- o derived type
- o device deviation
- o extension
- o feature
- o grouping
- o instance identifier

- o leaf-list
- o list
- o mandatory node
- o module
- o RPC
- o RPC operation
- o schema node
- o schema tree
- o state data
- o submodule
- o top-level data node
- o uses

The following terms are defined in [XML-INFOSET]:

- o attribute
- o document
- o document element
- o document type declaration (DTD)
- o element
- o information set
- o namespace

In the text, the following typographic conventions are used:

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

- o Other literal values are delimited by double quotes.

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

"a" DTD compatibility annotations [RNG-DTD];

"dc" Dublin Core metadata elements [RFC5013];

"dsrl" Document Semantics Renaming Language [DSRL];

"en" NETCONF event notifications [RFC5277];

"nc" NETCONF protocol [RFC4741];

"nma" NETMOD-specific schema annotations (see Section 5.3);

"nmf" NETMOD-specific XPath extension functions (see Section 12.7);

"rng" RELAX NG [RNG];

"sch" ISO Schematron [Schematron];

"xsd" W3C XML Schema [XSD].

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 |
| dsrl | http://purl.oclc.org/dsdl/dsrl |
| en | urn:ietf:params:xml:ns:netconf:notification:1.0 |
| nc | urn:ietf:params:xml:ns:netconf:base:1.0 |
| nma | urn:ietf:params:xml:ns:netmod:dSDL-annotations:1 |
| nmf | urn:ietf:params:xml:ns:netmod:xpath-extensions:1 |
| 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

2.1. Glossary of New Terms

- o ancestor datatype: Any datatype a given datatype is (transitively) derived from.
- o ancestor built-in datatype: The built-in datatype that is at the start of the type derivation chain for a given datatype.
- o hybrid schema: A RELAX NG schema with annotations, which embodies the same information as the source YANG module(s). See Section 8.1 for details.
- o implicit node: A data node that, if it is not instantiated in a data tree, may be added to the information set of that data tree (configuration, RPC input or output, notification) without changing the semantics of the data tree.

3. Objectives and Motivation

The main objective of this work is to complement YANG as a data modeling language with validation capabilities of DSDL schema languages, namely RELAX NG, Schematron and DSRL. This document describes the correspondence between grammatical, semantic and data type constraints expressed in YANG and equivalent DSDL patterns and rules. 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 may in principle be invertible, the inverse mapping from DSDL to YANG is beyond the scope of this document.

XML-based information models and XML-encoded data appear in several different forms in various phases of YANG data modeling and NETCONF workflow - configuration datastore contents, RPC requests and replies, and notifications. Moreover, RPC operations are characterized by an inherent diversity resulting from selective availability of capabilities and features. YANG modules can also define new RPC operations. The mapping should be able to accommodate this variability and generate schemas that are specifically tailored to a particular situation and thus considerably more effective for validation than generic all-encompassing schemas.

In order to cope with this variability, we assume that the DSDL schemas will be generated on demand for a particular purpose 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 an extended period of time in parallel to YANG modules.

The generated schemas are primarily intended as input to 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 the so-called hybrid schema, which is a single RELAX NG schema that describes grammatical constraints for the main data tree as well as for RPC operations and notifications. Semantic constraints and other information appearing in the input YANG modules is recorded in the hybrid schema in the form of foreign namespace annotations. The output of the first step can thus be considered a virtually complete equivalent of the input YANG modules.

2. In the second step, the hybrid schema from step 1 is transformed further to a coordinated set of fully conformant 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.

4. DSDL Schema Languages

Document Schema Definition Languages (DSDL) is a framework of schema languages that is being developed as the International Standard ISO/IEC 19757 [DSDL]. Unlike other approaches to XML document validation, most notably W3C XML Schema Definition (XSD) [XSD], the DSDL framework adheres to the principle of "small languages": Each of the DSDL constituents is a stand-alone schema language with a relatively narrow purpose and focus. Together, these schema languages may be used in a coordinated way to accomplish various validation tasks.

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

4.1. RELAX NG

RELAX NG (pronounced "relaxing") is an XML schema language for grammar-based validation and Part 2 of the ISO/IEC DSDL family of standards [RNG]. Like the W3C XML Schema language [XSD], it is able to describe constraints on the structure and contents of XML documents. However, unlike the DTD [XML] and XSD schema languages, RELAX NG intentionally avoids any infost 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 4.3).

As its base datatype library, RELAX NG uses the W3C XML Schema Datatype Library [XSD-D], 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 a few exceptions, such annotations may be placed anywhere in the schema and need no encapsulating elements such as `<xsd:annotation>` in XSD.

RELAX NG schemas can be represented in two equivalent syntaxes: XML and compact. The compact syntax is described in Annex C of the RELAX NG specification [RNG-CS], which was added to the standard in 2006 (Amendment 1). Automatic bidirectional conversions between the two syntaxes can be accomplished using several tools, for example Trang [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 usually work with the XML syntax. However, the compact syntax has two drawbacks:

- o 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 as many as four different syntactic constructs: documentation, grammar, initial and following annotations. Therefore, the impact of annotations on readability is often much stronger for the compact syntax than it is for the XML syntax.
- o In a computer program, it is more difficult to generate the compact syntax than the XML syntax. While a number of software libraries exist that make it easy to create an XML tree in the memory and then serialize it, no such aid is available for the compact syntax.

For these reasons, the mapping specification in this document uses 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".

4.2. Schematron

Schematron is Part 3 of DSDL that reached the status of a full ISO/IEC standard in 2006 [Schematron]. 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 components:

- o context - an XPath expression that defines the set of locations where the rule is to be applied;
- o assert or report condition - another XPath expression that is evaluated relative to the location matched by the context expression;
- o 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 [XPath] and Extensible Stylesheet Language Transformations (XSLT) [XSLT]. 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 the future).

Human-readable error messages are another feature that sets Schematron apart from other common schema languages. The messages may even contain XPath expressions that are evaluated in the actual context and thus refer to information items in the XML document being validated.

Another feature of Schematron that is used by the mapping are abstract patterns. These work essentially as macros and may also contain parameters which are supplied when the abstract pattern is used.

Schematron elements are qualified with namespace URI "http://purl.oclc.org/dsdl/schematron".

4.3. Document Semantics Renaming Language (DSRL)

DSRL (pronounced "disrule") is Part 8 of DSDL that reached the status of a full ISO/IEC standard in 2008 [DSRL]. Unlike RELAX NG and Schematron, DSRL is allowed to modify XML information set of the validated document. While DSRL is primarily intended for renaming XML elements and attributes, it can also define default values for XML attributes and default contents for XML elements or subtrees so that the default contents are inserted if they are missing in the validated documents. The latter feature is used by the YANG-to-DSDL mapping for representing YANG default contents consisting of leaf nodes with default values and their ancestor non-presence containers.

DSRL elements are qualified with namespace URI "http://purl.oclc.org/dsdl/dsrl".

5. Additional Annotations

Besides the DSDL schema languages, the mapping also uses three sets of annotations that are added as foreign-namespace attributes and elements 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. Although these data model items are not used for formal validation, they quite often carry important information for data model implementers. Therefore, they SHOULD be included in the hybrid schema and MAY also appear in the final validation schemas.

The third set are NETMOD-specific annotations. They are specifically designed for the hybrid schema and convey semantic constraints and other information that cannot be expressed directly in RELAX NG. In the second mapping step, these annotations are converted to Schematron and DSRL rules.

5.1. 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 from [RFC5013].

Dublin Core elements are qualified with namespace URI "http://purl.org/dc/terms".

5.2. RELAX NG DTD Compatibility Annotations

DTD compatibility annotations are a part of the RELAX NG DTD Compatibility specification [RNG-DTD]. 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 formatted by several RELAX NG tools.

DTD compatibility annotations are qualified with namespace URI "http://relaxng.org/ns/compatibility/annotations/1.0".

5.3. 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" which appear in various locations of the hybrid schema. YANG statements are mapped to these annotations in a straightforward way. In most cases, the annotation attributes and elements have the same name as the corresponding YANG statement.

Table 2 lists alphabetically the names of NETMOD-specific annotation attributes (prefixed with "@") and elements (in angle brackets) along with a reference to the section where their use is described. Appendix A contains a RELAX NG schema for this annotation vocabulary.

| annotation | section | note |
|---------------------------|--------------------|------|
| @nma:config | 10.9 | |
| <nma:data> | 8.1 | 4 |
| @nma:default | 10.12 | |
| <nma:error-app-tag> | 10.16 | 1 |
| <nma:error-message> | 10.17 | 1 |
| @nma:if-feature | 10.22 | |
| @nma:implicit | 10.11, 10.7, 10.12 | |
| <nma:input> | 8.1 | 4 |
| <nma:instance-identifier> | 10.53.7 | 2 |
| @nma:key | 10.26 | |
| @nma:leaf-list | 10.28 | |
| @nma:leafref | 10.53.8 | |
| @nma:mandatory | 10.8 | |
| @nma:max-elements | 10.28 | |
| @nma:min-elements | 10.28 | |

| | | |
|---------------------|-------|---|
| @nma:module | 10.34 | |
| <nma:must> | 10.35 | 3 |
| <nma:notification> | 8.1 | 4 |
| <nma:notifications> | 8.1 | 4 |
| @nma:ordered-by | 10.38 | |
| <nma:output> | 8.1 | 4 |
| <nma:rpc> | 8.1 | 4 |
| <nma:rpcs> | 8.1 | 4 |
| @nma:status | 10.51 | |
| @nma:unique | 10.55 | |
| @nma:units | 10.56 | |
| @nma:when | 10.59 | |

Table 2: NETMOD-specific annotations

Notes:

1. Appears only as a 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>.
4. Marker element in the hybrid schema.

6. Overview of the Mapping

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

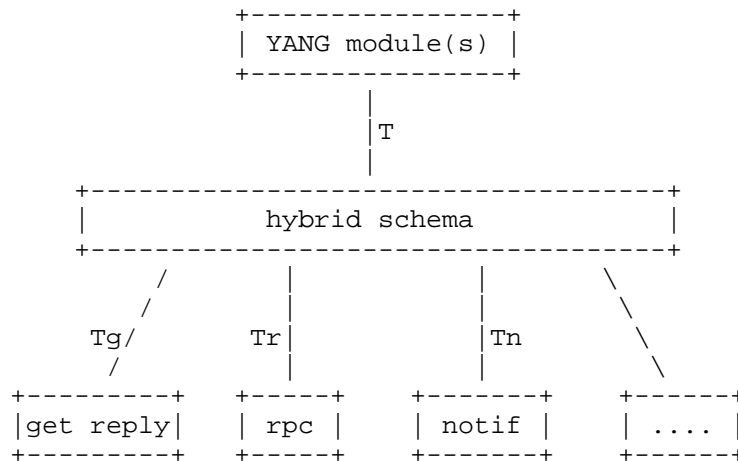


Figure 1: Structure of the mapping

The mapping procedure is divided into two steps:

1. Transformation T in the first step maps one or more YANG modules to the hybrid schema (see Section 8.1). 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 foreign-namespaces annotations.
2. In the second step, the hybrid schema may be 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 shows three simple possibilities as examples. In the process, appropriate parts of the hybrid schema are extracted and specific annotations transformed to equivalent, but usually more complex, Schematron patterns, DSRL element maps etc.

An implementation of the mapping algorithm MUST accept 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 root nodes of the datastore hierarchy. 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 input modules import (directly or transitively).

Other information contained in input YANG modules, such as semantic constraints and default values, are recorded in the hybrid schema as annotations - XML attributes or elements qualified with namespace URI "urn:ietf:params:xml:ns:netmod:dSDL-annotations:1". Metadata describing the YANG modules are mapped to Dublin Core annotations elements (Section 5.1). Finally, documentation strings are mapped to <a:documentation> elements belonging to the DTD compatibility vocabulary (Section 5.2).

The output of the second step is a coordinated set of three DSDL schemas corresponding to a specific data object and context:

- o RELAX NG schema describing the grammatical and datatype constraints;
- o Schematron schema expressing other constraints such as uniqueness of list keys or user-specified semantic rules;
- o DSRL schema containing the specification of default contents.

7. NETCONF Content Validation

This section describes how the schemas generated by the YANG-to-DSDL mapping are supposed to be applied for validating XML instance documents such as the contents of a datastore or various NETCONF messages.

The validation proceeds in the following steps, which are also illustrated in Figure 2:

1. The XML instance document is checked for grammatical and data type validity using the RELAX NG schema.
2. Default values for leaf nodes have to be applied and their ancestor containers added where necessary. It is important to add the implicit nodes before the next validation step because YANG specification [RFC6020] requires 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 validated XML document.
3. The semantic constraints are checked using the Schematron schema.

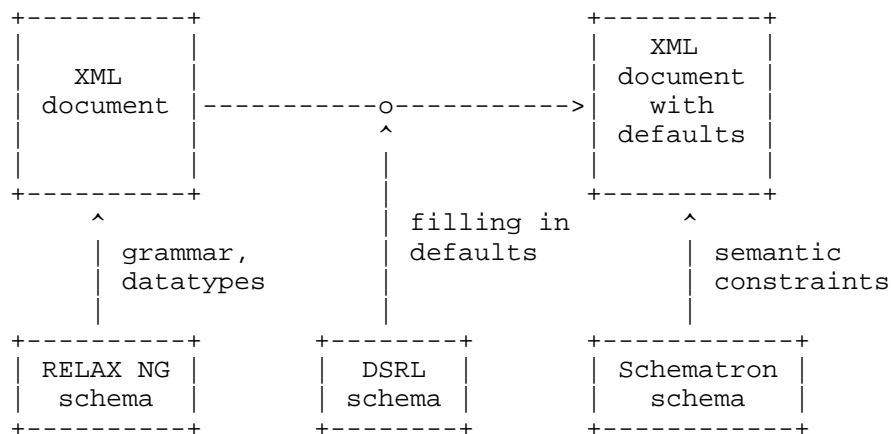


Figure 2: Outline of the validation procedure

8. Design Considerations

YANG data models could in principle be mapped to the 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.

8.1. Hybrid Schema

As was explained in Section 6, the first step of the mapping produces an intermediate document - the hybrid schema, which specifies all constraints for the entire data model in a single RELAX NG schema.

Every input YANG module corresponds to exactly one embedded grammar in the hybrid schema. This separation of input YANG modules allows each embedded grammar to include named pattern definitions into its own namespace, which is important for mapping YANG groupings (see Section 9.2 for additional details).

In addition to grammatical and datatype constraints, YANG modules provide other important information that cannot be expressed in a RELAX NG schema: semantic constraints, default values, metadata, documentation and so on. Such information items are represented in the hybrid schema as XML attributes and elements belonging to the namespace with the following URI:
"urn:ietf:params:xml:ns:netmod:dSDL-annotations:1". A complete list of these annotations is given in Section 5.3, detailed rules about their use are then contained in the following sections.

YANG modules define data models not only for configuration and state data but also for (multiple) RPC operations [RFC4741] and/or event notifications [RFC5277]. In order to be able to capture all three types of data models in one schema document, the hybrid schema uses special markers that enclose sub-schemas for configuration and state data, individual RPC operations (both input and output part) and individual notifications.

The markers are the following XML elements in the namespace of NETMOD-specific annotations (URI
urn:ietf:params:xml:ns:netmod:dSDL-annotations:1):

| Element name | Role |
|-------------------|---------------------------------------|
| nma:data | encloses configuration and state data |
| nma:rpcs | encloses all RPC operations |
| nma:rpc | encloses an individual RPC operation |
| nma:input | encloses an RPC request |
| nma:output | encloses an RPC reply |
| nma:notifications | encloses all notifications |
| nma:notification | encloses an individual notification |

Table 3: Marker elements in the hybrid schema

For example, consider a data model formed by two YANG modules "example-a" and "example-b" that define nodes in the namespaces "http://example.com/ns/example-a" and "http://example.com/ns/example-b". Module "example-a" defines configuration/state data, RPC methods and notifications, whereas "example-b" defines only configuration/state data. The hybrid schema can then be schematically represented as follows:

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:nma="urn:ietf:params:xml:ns:netmod:dSDL-annotations:1"
  xmlns:exa="http://example.com/ns/example-a"
  xmlns:exb="http://example.com/ns/example-b"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <grammar nma:module="example-a"
      ns="http://example.com/ns/example-a">
      <start>
        <nma:data>
          ...configuration and state data defined in "example-a"...
        </nma:data>
        <nma:rpcs>
          <nma:rpc>
            <nma:input>
              <element name="exa:myrpc">
                ...
              </element>
            </nma:input>
            <nma:output>
```



```

        ...
        </nma:output>
    </nma:rpc>
    ...
</nma:rpcs>
<nma:notifications>
    <nma:notification>
        <element name="exa:mynotif">
            ...
            </element>
        </nma:notification>
    ...
</nma:notifications>
</start>
...local named pattern definitions of example-a...
</grammar>
<grammar nma:module="example-b"
    ns="http://example.com/ns/example-a">
    <start>
        <nma:data>
            ...configuration and state data defined in "example-b"...
        </nma:data>
        <nma:rpcs/>
        <nma:notifications/>
    </start>
    ...local named pattern definitions of example-b...
</grammar>
</start>
...global named pattern definitions...
</grammar>

```

A complete hybrid schema for the data model of a DHCP server is given in Appendix C.2.

8.2. Modularity

Both YANG and RELAX NG offer means for modularity, i.e., for splitting the contents of a full schema into separate modules 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 MIB modules. The following differences are important:

- o 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 data tree from module B 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.

- o The names of imported YANG groupings and typedefs are qualified with the namespace of the imported module. On the other hand, the names of data nodes contained inside the imported groupings, when used within the importing module, become part of the importing module's 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 definitions of named patterns must be included from an external schema which has to be prepared in a special way (see [Vli04], Chapter 11).

In order to map, as much as possible, the modularity of YANG to RELAX NG, a validating RELAX NG schema (the result of the second mapping step) has to be split into two files, one of them containing all global definitions that are mapped from top-level YANG groupings appearing in all input YANG module. This RELAX NG schema MUST NOT define any namespace via the @ns attribute.

The other RELAX NG schema file then defines actual data trees mapped from input YANG modules, each of them enclosed in an own embedded grammar. Those embedded grammars in which at least one of the global definitions is used MUST include the first schema with definitions and also MUST define the local namespace using the @ns attribute. This way, the global definitions can be used inside different embedded grammar, each time accepting a different local namespace.

Named pattern definition that are mapped from non-top-level YANG groupings MUST be placed inside the embedded grammar corresponding to the YANG module where the grouping is defined.

In the hybrid schema, we need to distinguish the global and non-global named pattern definitions while still keeping the hybrid schema in one file. This is accomplished in the following way:

- o Every global definition MUST be placed as a child of the the outer <rng:grammar> element (the document root of the hybrid schema).
- o Every non-global definitions MUST be placed as a child of the corresponding embedded <rng:grammar> element.

YANG also allows for splitting a module into a number of submodules. However, as submodules have no impact on the scope of identifiers and

namespaces, the modularity based on submodules is not mapped in any way. The contents of submodules is therefore handled as if the submodule text appeared directly in the main module.

8.3. Granularity

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 corresponds to a named pattern definition. 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 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 is not feasible to map YANG's powerful extension mechanisms to those available in RELAX NG. For this reason, the mapping essentially keeps the granularity of the original YANG data model: YANG groupings and definitions of derived types usually have direct counterparts in definitions of named patterns in the resulting RELAX NG schema.

8.4. Handling of XML Namespaces

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 allows for multiple input YANG modules, which 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 the hybrid schema and validation RELAX NG schemas generated in the second step, the namespaces MUST be declared as follows:

1. The root <rng:grammar> MUST have @xmlns:xxx attributes declaring prefixes of all namespaces that are used in the data model. The prefixes SHOULD be identical to those defined in the 'prefix' statements. An implementation of the mapping MUST resolve all collisions in the prefixes defined by different input modules, if there are any.
2. Each embedded <rng:grammar> element MUST declare the namespace of the corresponding module using the @ns attribute. This way, the names of nodes defined by global named patterns are able to adopt the local namespace of each embedded grammar, as explained in Section 8.2.

This setup is illustrated by the example at the end of Section 8.1.

DSRL schemas may declare any number of target namespaces via the standard XML attributes xmlns:xxx.

In contrast, Schematron requires all used namespaces to be defined in the <sch:ns> subelements of the document element <sch:schema>.

9. Mapping YANG Data Models to the Hybrid Schema

This section explains the main principles governing the first step of the mapping. Its result is the hybrid schema which is described in Section 8.1.

A detailed specification of the mapping of individual YANG statements is contained in the following Section 10.

9.1. Occurrence Rules for Data Nodes

In DSDL schema languages, occurrence constraints for a node are always localized together with that node. In a RELAX NG schema, for example, `<rng:optional>` pattern appears as the parent element of the pattern defining a leaf or non-leaf element. Similarly, DSRL specifies default contents separately for every single node, be it a leaf or non-leaf element.

For leaf nodes in YANG modules, the occurrence constraints are also easily inferred from the substatements of 'leaf'. On the other hand, for a YANG container it is often necessary to examine its entire subtree in order to determine the container's occurrence constraints.

Therefore, one of the goals of the first mapping step is to infer the occurrence constraints for all data nodes and mark accordingly the corresponding `<rng:element>` patterns in the hybrid schema so that any transformation procedure in the second mapping step can simply use this information and need not examine the subtree again.

First, it has to be decided whether a given data node must always be present in a valid configuration. If so, such a node is called mandatory, otherwise it is called optional. This constraint is closely related to the notion of mandatory nodes in Section 3.1 in [RFC6020]. The only difference is that this document also considers list keys to be mandatory.

The other occurrence constraint has to do with the semantics of the 'default' statement and the possibility of removing empty non-presence containers. As a result, the information set of a valid configuration may be modified by adding or removing certain leaf or container elements without changing the meaning of the configuration. In this document, such elements are called implicit. In the hybrid schema, they can be identified as RELAX NG patterns having either `@nma:default` or `@nma:implicit` attribute.

Note that both occurrence constraints apply to containers at the top level of the data tree, and then also to other containers under the additional condition that their parent node exists in the instance

document. For example, consider the following YANG fragment:

```
container outer {
  presence 'Presence of "outer" means something.';
  container c1 {
    leaf foo {
      type uint8;
      default 1;
    }
  }
  container c2 {
    leaf-list bar {
      type uint8;
      min-elements 0;
    }
  }
  container c3 {
    leaf baz {
      type uint8;
      mandatory true;
    }
  }
}
```

Here, container "outer" has the 'presence' substatement, which means that it is optional and not implicit. If "outer" is not present in a configuration, its child containers are not present as well. However, if "outer" does exist, it makes sense to ask which of its child containers are optional and which are implicit. In this case, "c1" is optional and implicit, "c2" is optional but not implicit and "c3" is mandatory (and therefore not implicit).

The following subsections give precise rules for determining whether a container is optional or mandatory and whether it is implicit. In order to simplify the recursive definition of these occurrence characteristics, it is useful to define them also for other types of YANG schema nodes, i.e., leaf, list, leaf-list and anyxml and choice.

9.1.1.1. Optional and Mandatory Nodes

The decision whether a given node is mandatory or optional is governed by the following rules:

- o Leaf, anyxml and choice nodes are mandatory if they contain the substatement "mandatory true;". For a choice node this means that at least one node from exactly one case branch must exist.

- o In addition, a leaf node is mandatory if it is declared as a list key.
- o A list or leaf-list node is mandatory if it contains the 'min-elements' substatement with an argument value greater than zero.
- o A container node is mandatory if its definition does not contain the 'presence' substatement and at least one of its child nodes is mandatory.

A node which is not mandatory is said to be optional.

In RELAX NG, definitions of nodes that are optional must be explicitly wrapped in the `<rng:optional>` element. The mapping MUST use the above rules to determine whether a YANG node is optional and if so, insert the `<rng:optional>` element in the hybrid schema.

However, alternatives in `<rng:choice>` MUST NOT be defined as optional in the hybrid schema. If a choice in YANG is not mandatory, `<rng:optional>` MUST be used to wrap the entire `<rng:choice>` pattern.

9.1.2. Implicit Nodes

The following rules are used to determine whether a given data node is implicit:

- o List, leaf-list and anyxml nodes are never implicit.
- o A leaf node is implicit if and only if it has a default value, defined either directly or via its datatype.
- o A container node is implicit if and only if it does not have the 'presence' substatement, none of its children are mandatory and at least one child is implicit.

In the hybrid schema, all implicit containers, as well as leafs that obtain their default value from a typedef and don't have the `@nma:default` attribute, MUST be marked with `@nma:implicit` attribute having the value of "true".

Note that Section 7.9.3 in [RFC6020] specifies other rules that must be taken into account when deciding whether a given container or leaf appearing inside a case of a choice is ultimately implicit or not. Specifically, a leaf or container under a case can be implicit only if the case appears in the argument of the choice's 'default' statement. However, this is not sufficient by itself but also depends on the particular instance XML document, namely on the presence or absence of nodes from other (non-default) cases. The

details are explained in Section 11.3.

9.2. Mapping YANG Groupings and Typedefs

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 Section 5.5 in [RFC6020]. Second, top-level symbols from external modules may be 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:

- o 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 ("__").
- o 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.
- o Finally, since the names of groupings and typedefs in YANG have different namespaces, an additional underscore character is added to the beginning of the mangled names of all groupings.

An additional complication is caused by the YANG rules for subelement ordering (see, e.g., Section 7.5.7 in [RFC6020]): In RPC input and output parameters, subelements must follow the order specified in the data model, otherwise the order is arbitrary. Consequently, if a grouping is used both in RPC input/output parameters and elsewhere, it MUST be mapped to two different named pattern definitions - one with fixed order and the other with arbitrary order. To distinguish them, the "__rpc" suffix MUST be appended to the version with fixed order.

EXAMPLE. Consider the following YANG module which imports the standard module "ietf-inet-types" [RFC6021]:


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

The hybrid schema generated by the first mapping step will then contain the following two (global) named pattern definitions:

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

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

9.2.1. 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 provides two substatements of 'uses' which modify expansions of groupings:

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

- o 'augment' statement can be used for adding new schema nodes to the grouping contents.

Both 'refine' and 'augment' statements are quite powerful in that they can address, using XPath-like expressions as their arguments, schema nodes that are arbitrarily deep inside the grouping contents. In contrast, modifications of named pattern definitions in RELAX NG are applied exclusively at the topmost level of the named pattern contents. In order to achieve a modifiability of named patterns comparable to YANG, a RELAX NG schema would have to be extremely flat (cf. Section 8.3) 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 replaced by the contents 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 contents, 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 hybrid schema contains three global named pattern definitions corresponding to the three groupings, namely

```
<rng:define name="_example2__leaves">
  <rng:interleave>
    <rng:ref name="_example2__fr"/>
    <rng:ref name="_example2__es"/>
  </rng:interleave>
</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 hybrid schema is a single named pattern reference:

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

Now assume that the "uses leaves" statement contains a 'refine' substatement, for example:

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

The resulting hybrid schema now contains just one named pattern definition - "_example2__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 hybrid schema now looks like this:

```
<nma:data>
  <rng:interleave>
    <rng:ref name="_example2__fr"/>
    <rng:optional>
      <rng:element name="ex2:hoja" nma:default="alamo">
        <rng:data type="string"/>
      </rng:element>
    </rng:optional>
  </rng:interleave>
</nma:data>
```

9.2.2. Type Derivation Chains

RELAX NG has no equivalent of the type derivation mechanism in YANG that allows to restrict a built-in type (perhaps in multiple steps) by adding new constraints. Whenever a derived YANG type is used without restrictions - as a substatement of either 'leaf' or another 'typedef' - then the 'type' statement is mapped simply to a named pattern reference <rng:ref>, and the type definition is mapped to a RELAX NG named pattern definition <rng:define>. However, if any restrictions are specified as substatements of the 'type' statement, the type definition MUST be expanded at that point so that only the ancestor built-in type appears in the hybrid schema, restricted with facets that 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" has no restrictions and is therefore mapped simply to the reference <rng:ref name="example3__dozen"/> and the corresponding named pattern is defined as follows:

```
<rng:define name="example3__dozen">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">1</rng:param>
    <rng:param name="maxInclusive">12</rng: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 will not contain any named pattern definition and the leaf "month" will be mapped directly to

```
<rng:element name="ex3:month">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">7</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:element>
```

The mapping of type derivation chains may be further complicated by the presence of the 'default' statement in type definitions. In the simple case, when a type definition containing the 'default' statement is used without restrictions, the 'default' statement is mapped to the @nma:default attribute attached to the <rng:define> element.

However, if that type definition has to be expanded due to restrictions, the @nma:default annotation arising from the expanded type or ancestor types in the type derivation chain MUST be attached to the pattern where the expansion occurs. If there are multiple 'default' statements in consecutive steps of the type derivation, only the 'default' statement that is closest to the expanded type is used.

EXAMPLE. Consider this variation of the last example:

```
module example3bis {
  namespace "http://example.com/ns/example3bis";
  prefix ex3bis;
  typedef dozen {
    type uint8 {
      range 1..12;
    }
    default 7;
  }
  leaf month {
    type dozen;
  }
}
```

The 'typedef' statement in this module is mapped to the following named pattern definition:

```
<rng:define name="example3bis__dozen" @nma:default="7">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">1</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:define>
```

If the "dozen" type is restricted when used in the leaf "month" definition as in the previous example, the "dozen" type has to be expanded and @nma:default becomes an attribute of the <ex3bis:month> element definition:

```
<rng:element name="ex3bis:month" @nma:default="7">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">7</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:element>
```

However, if the definition of the leaf "month" itself contained the 'default' substatement, the default specified for the "dozen" type would be ignored.

9.3. Translation of XPath Expressions

YANG uses full XPath 1.0 syntax [XPath] for the arguments of 'must', 'when' and 'path' statements. As the names of data nodes defined in a YANG module always belong to the namespace of that YANG module, YANG adopted a simplification similar to the concept of default namespace in XPath 2.0: node names in XPath expressions needn't carry a namespace prefix inside the module where they are defined and the

local module's namespace is assumed.

Consequently, all XPath expressions MUST be translated into a fully conformant XPath 1.0 expression: Every unprefix node name MUST be prepended with the local module's namespace prefix as declared by the 'prefix' statement.

XPath expressions appearing inside top-level groupings require special attention because all unprefix node names contained in them must adopt the namespace of each module where the grouping is used (cf. Section 8.2. In order to achieve this, the local prefix MUST be represented using the variable "\$pref" in the hybrid schema. A Schematron schema which encounters such an XPath expression then supplies an appropriate value for this variable via a parameter to an abstract pattern to which the YANG grouping is mapped (see Section 11.2).

For example, XPath expression "/dhcp/max-lease-time" appearing in a YANG module with the "dhcp" prefix will be translated to

- o "\$pref:dhcp/\$pref:max-lease-time", if the expression is inside a top-level grouping;
- o "dhcp:dhcp/dhcp:max-lease-time", otherwise.

YANG also uses other XPath-like expressions, namely key identifiers and "descendant schema node identifiers" (see the ABNF production for and "descendant-schema-nodeid" in Section 12 of [RFC6020]). These expressions MUST be translated by adding local module prefixes as well.

9.4. YANG Language Extensions

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 they can be used as the standard YANG statements, from which they are distinguished by 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 any 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, see Section 11.1 in [RFC6020].

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

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

10. Mapping YANG Statements to the Hybrid Schema

Each subsection in this section is devoted to one YANG statement and provides the specification of how the statement is mapped to the hybrid schema. 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 procedure 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. Any changes to this default recursive procedure are explicitly specified.

YANG XML encoding rules translate to the following rules for ordering multiple subelements:

1. Within the `<nma:rpcs>` subtree (i.e., for input and output parameters of an RPC operation) the order of subelements is fixed and their definitions in the hybrid schema **MUST** follow the order specified in the source YANG module.
2. When mapping the 'list' statement, all keys **MUST** come before any other subelements and in the same order as they are declared in the 'key' statement. The order of the remaining (non-key) subelements is not specified, so their definitions in the hybrid schema **MUST** be enclosed in the `<rng:interleave>` element.
3. Otherwise, the order of subelements is arbitrary and, consequently, all definitions of subelements in the hybrid schema **MUST** be enclosed in the `<rng:interleave>` element.

The following conventions are used in this section:

- o The argument of the statement being mapped is denoted by `ARGUMENT`.
- o 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

This statement is mapped to `<rng:element>` element and `ARGUMENT` with prepended local namespace prefix becomes the value of its `@name` attribute. The contents of `<rng:element>` are

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

Substatements of the 'anyxml' statement, if any, MAY be mapped to additional children of the <rng:element> element.

If at least one '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 root <rng:grammar> element (cf. [Vli04], 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 in a module with namespace prefix "yam"

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

is mapped to the following fragment:

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

An anyxml node is optional if there is no "mandatory true;" substatement. The <rng:element> element then MUST be wrapped in <rng:optional>, except when the 'anyxml' statement is a child of the 'choice' statement and thus forms a shorthand case for that choice (see Section 9.1.1 for details).

10.2. The 'argument' Statement

This statement is not mapped to the output schema, but see the rules for handling extensions in Section 9.4.

10.3. The 'augment' Statement

As a substatement of 'uses', this statement is handled as a part of 'uses' mapping, see Section 10.57.

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 augmented 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

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 10.53.6.

10.5. The 'belongs-to' Statement

This statement is not used since the processing of submodules is always initiated from the main module, see Section 10.24.

10.6. The 'bit' Statement

This statement is handled within the "bits" type, see Section 10.53.4.

10.7. The 'case' Statement

This statement is mapped to <rng:group> or <rng:interleave> element, depending on whether the statement belongs to an definition of an RPC operation or not. If the argument of a sibling 'default' statement equals to ARGUMENT, @nma:implicit attribute with the value of "true" MUST be added to that <rng:group> or <rng:interleave> element. The @nma:implicit attribute MUST NOT be used for nodes at the top-level of a non-default case (see Section 7.9.3 in [RFC6020]).

10.8. The 'choice' Statement

This statement is mapped to <rng:choice> element.

If 'choice' has the 'mandatory' substatement with the value of "true", the attribute @nma:mandatory MUST be added to the <rng:choice> element with the value of ARGUMENT. This case may require additional handling, see Section 11.2.1. Otherwise, if "mandatory true;" is not present, the <rng:choice> element MUST be wrapped in <rng:optional>.

The alternatives in `<rng:choice>` - mapped from either the 'case' statement or a shorthand case - MUST NOT be defined as optional.

10.9. The 'config' Statement

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

10.10. The 'contact' Statement

This statement SHOULD NOT be used by the mapping since the hybrid schema may be mapped from multiple YANG modules created by different authors. The hybrid schema contains references to all input modules in the Dublin Core elements `<dc:source>`, see Section 10.34. The original YANG modules are the authoritative sources of the authorship information.

10.11. The 'container' Statement

Using the rules specified in Section 9.1.1, 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 with prepended local namespace prefix as the value of its `@name` attribute.

Finally, using the rules specified in Section 9.1.2, the mapping algorithm MUST determine whether the container is implicit, and if so, add the attribute `@nma:implicit` with the value of "true" to the `<rng:element>` element.

10.12. The 'default' Statement

If this statement is a substatement of 'leaf', it is mapped to the `@nma:default` attribute of PARENT and ARGUMENT becomes its value.

As a substatement of 'typedef', the 'default' statement is also mapped to the `@nma:default` attribute with the value of ARGUMENT. The placement of this attribute depends on whether or not the type definition has to be expanded when it is used:

- o If the type definition is not expanded, `@nma:default` becomes an attribute of the `<rng:define>` pattern resulting from the parent 'typedef' mapping.

- o Otherwise, @nma:default becomes an attribute of the ancestor RELAX NG pattern inside which the expansion takes place.

Details and an example are given in Section 9.2.2.

Finally, as a substatement of 'choice', the 'default' statement identifies the default case and is handled within the 'case' statement, see Section 10.7. If the default case uses the shorthand notation where the 'case' statement is omitted, the @nma:implicit attribute with the value of "true" is either attached to the node representing the default case in the shorthand notation or, alternatively, an extra <rng:group> element MAY be inserted and the @nma:implicit attribute attached to it. In the latter case, the net result is the same as if the 'case' statement wasn't omitted for the default case.

EXAMPLE. The following 'choice' statement in a module with namespace prefix "yam"

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

is either mapped directly to

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

or the default case may be wrapped in an extra <rng:group>:

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

10.13. The 'description' Statement

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 'deviation' Statement

This statement is ignored. However, it is assumed that all deviations are known beforehand and the corresponding changes have already been applied to the input YANG modules.

10.15. The 'enum' Statement

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 annotation elements, see [RNG], section 6.

10.16. The 'error-app-tag' Statement

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 10.35.

10.17. The 'error-message' Statement

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 10.35.

10.18. The 'extension' Statement

This statement is ignored. However, extensions to the YANG language MAY be mapped as described in Section 9.4.

10.19. The 'feature' Statement

This statement is ignored.

10.20. 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 9.2.

As explained in Section 8.2, a named pattern definition MUST be placed

- o as a child of the root `<rng:grammar>` element if the corresponding grouping is defined at the top level of an input YANG module;
- o otherwise as a child of the embedded `<rng:grammar>` element corresponding to the module in which the grouping is defined.

Whenever a grouping is used with refinements and/or augments, it is expanded so that the refinements and augments may be applied in place to the prescribed schema nodes. See Section 9.2.1 for further details and an example.

An implementation MAY offer the option of mapping all 'grouping' statements as named pattern definitions in the output RELAX NG schema even if they are not referenced. This is useful for mapping YANG "library" modules that typically contain only 'typedef' and/or 'grouping' statements.

10.21. The 'identity' Statement

This statement is mapped to the following named pattern definition which is placed as a child of the root `<rng:grammar>` element:

```
<rng:define name="__PREFIX_ARGUMENT">
  <rng:choice>
    <rng:value type="QName">PREFIX:ARGUMENT</rng:value>
    <rng:ref name="IDENTITY1"/>
    ...
  </rng:choice>
</rng:define>
```

where

PREFIX is the prefix used in the hybrid schema for the namespace of the module where the current identity is defined.

IDENTITY1 is the name of of the named pattern corresponding to an identity which is derived from the current identity. Exactly one `<rng:ref>` element MUST be present for every such identity.

EXAMPLE ([RFC6020], Section 7.16.3). The identities in the input 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";
  }
}
```

will be mapped to the following named pattern definitions:

```
<define name="__crypto_crypto-alg">
  <choice>
    <value type="QName">crypto:crypto-alg</value>
    <ref name="__des_des"/>
    <ref name="__des_des3"/>
  </choice>
</define>
<define name="__des_des">
  <value type="QName">des:des</value>
</define>
<define name="__des_des3">
  <value type="QName">des:des3</value>
</define>
```

10.22. The 'if-feature' Statement

ARGUMENT together with arguments of all sibling 'if-feature' statements (with added prefixes, if missing) MUST be collected in a space-separated list which becomes the value of the @nma:if-feature attribute. This attribute is attached to PARENT.

10.23. The 'import' Statement

This statement is not specifically mapped. The module whose name is in ARGUMENT has to be parsed so that the importing module is able to use its top-level groupings, typedefs and identities, 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.

10.24. 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 appeared directly in 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.

10.25. The 'input' Statement

This statement is handled within 'rpc' statement, see Section 10.50.

10.26. The 'key' Statement

This statement is mapped to @nma:key attribute. ARGUMENT 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.

10.27. The 'leaf' Statement

This statement is mapped to the <rng:element> element and ARGUMENT with prepended local namespace prefix becomes the value of its @name attribute.

If the leaf is optional, i.e., if there is no "mandatory true;" substatement and the leaf is not declared among the keys of an enclosing list, then the <rng:element> element MUST be enclosed in <rng:optional>, except when the 'leaf' statement is a child of the 'choice' statement and thus represents a shorthand case for that choice (see Section 9.1.1 for details).

10.28. The 'leaf-list' Statement

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 then added as a child element of PARENT and ARGUMENT with prepended local namespace prefix becomes the value of its @name attribute. Another attribute, @nma:leaf-list, MUST also be added to this `<rng:element>` element with the value of "true". 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. A leaf-list appearing in a module with the namespace prefix "yam"

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

10.29. The 'length' Statement

This statement is handled within the "string" type, see Section 10.53.10.

10.30. The 'list' Statement

This statement is mapped exactly as the 'leaf-list' statement, see Section 10.28. The only difference is that the @nma:leaf-list annotation either MUST NOT be present or MUST have the value of "false".

When mapping the substatements of 'list', the order of children of the list element MUST be specified so that list keys, if there are any, always appear in the same order as they are defined in the 'key' substatement and before other children, see [RFC6020], Section 7.8.5. In particular, if a list key is defined in a grouping but the list node itself is not a part of the same grouping, and the position of the 'uses' statement would violate the above ordering requirement, the grouping MUST be expanded, i.e., the 'uses' statement replaced by the grouping contents.

For example, consider the following YANG fragment of a module with the prefix "yam":

```
grouping keygrp {
  leaf clef {
    type uint8;
  }
}
list foo {
  key clef;
  leaf bar {
    type string;
  }
  leaf baz {
    type string;
  }
  uses keygrp;
}
```

is mapped to the following RELAX NG fragment:

```
<rng:zeroOrMore>
  <rng:element name="yam:foo" nma:key="yam:clef">
    <rng:element name="yam:clef">
      <rng:data type="unsignedByte"/>
    </rng:element>
    <rng:interleave>
      <rng:element name="yam:bar">
        <rng:data type="string"/>
      </rng:element>
      <rng:element name="yam:baz">
        <rng:data type="string"/>
      </rng:element>
    </rng:interleave>
  </rng:element>
</rng:zeroOrMore>
```

Note that the "keygrp" grouping is expanded and the definition of "yam:clef" is moved before the <rng:interleave> pattern.

10.31. The 'mandatory' Statement

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 9.1.1.

As a substatement of 'choice', this statement is also mapped to the @nma:mandatory attribute which is added to PARENT. The value of this attribute is the argument of the parent 'choice' statement.

10.32. The 'max-elements' Statement

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.28.

10.33. The 'min-elements' Statement

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.28.

10.34. The 'module' Statement

This statement is mapped to an embedded <rng:grammar> pattern having the @nma:module attribute with the value of ARGUMENT. In addition, a <dc:source> element SHOULD be created as a child of this <rng:grammar> element and contain ARGUMENT as a metadata reference to the input YANG module. See also Section 10.49.

Substatements of the 'module' statement MUST be mapped so that

- o statements representing configuration/state data are mapped to descendants of the <nma:data> element;
- o statements representing the contents of RPC requests or replies are mapped to descendants of the <nma:rpcs> element;
- o statements representing the contents of event notifications are mapped to descendants of the <nma:notifications> element.

10.35. The 'must' Statement

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 9.3). The <nma:must> element may have other subelements resulting from mapping the 'error-app-tag' and 'error-message' substatements. Other substatements of 'must', i.e., 'description' and 'reference', are ignored.

EXAMPLE. YANG statement in the "dhcp" module

```
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>
```

10.36. The 'namespace' Statement

This statement is mapped simultaneously in two ways:

1. To the @xmlns:PREFIX attribute of the root <rng:grammar> element where PREFIX is the namespace prefix specified by the sibling 'prefix' statement. ARGUMENT becomes the value of this attribute.
2. To the @ns attribute of PARENT, which is an embedded <rng:grammar> pattern. ARGUMENT becomes the value of this attribute.

10.37. The 'notification' Statement

This statement is mapped to the following subtree of the <nma:notifications> element in the hybrid schema (where PREFIX is the prefix of the local YANG module):

```
<nma:notification>
  <rng:element name="PREFIX:ARGUMENT">
    ...
  </rng:element>
</nma:notification>
```

Substatements of 'notification' are mapped under <rng:element name="PREFIX:ARGUMENT">.

10.38. The 'ordered-by' Statement

This statement is mapped to @nma:ordered-by attribute and ARGUMENT becomes the value of this attribute. See Section 10.28 for an example.

10.39. The 'organization' Statement

This statement is ignored by the mapping because the hybrid schema may be mapped from multiple YANG modules authored by different parties. The hybrid schema SHOULD contain references to all input modules in the Dublin Core <dc:source> elements, see Section 10.34. The original YANG modules are the authoritative sources of the authorship information.

10.40. The 'output' Statement

This statement is handled within the 'rpc' statement, see Section 10.50.

10.41. The 'path' Statement

This statement is handled within the "leafref" type, see Section 10.53.8.

10.42. The 'pattern' Statement

This statement is handled within the "string" type, see Section 10.53.10.

10.43. The 'position' Statement

This statement is ignored.

10.44. The 'prefix' Statement

This statement is handled within the sibling 'namespace' statement, see Section 10.36, or within the parent 'import' statement, see Section 10.23. As a substatement of 'belongs-to' (in submodules), the 'prefix' statement is ignored.

10.45. The 'presence' Statement

This statement influences the mapping of the parent container (Section 10.11): the parent container definition MUST be wrapped in `<rng:optional>`, regardless of its contents. See also Section 9.1.1.

10.46. The 'range' Statement

This statement is handled within numeric types, see Section 10.53.9.

10.47. The 'reference' Statement

This statement is mapped to `<a:documentation>` element and its text is set to ARGUMENT prefixed with "See: ".

10.48. The 'require-instance' Statement

This statement is handled within "instance-identifier" type (Section 10.53.7).

10.49. The 'revision' Statement

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 [RFC6020]. This date SHOULD be recorded, together with the name of the YANG module, in the corresponding Dublin Core `<dc:source>` element (see Section 10.34), for example in this form:

```
<dc:source>YANG module 'foo', revision 2010-03-02</dc:source>
```

The 'description' substatement of 'revision' is ignored.

10.50. The 'rpc' Statement

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

```
<nma:rpc>
  <nma:input>
    <rng:element name="PREFIX:ARGUMENT">
      ... mapped contents of 'input' ...
    </rng:element>
  </nma:input>
  <nma:output">
    ... mapped contents of 'output' ...
  </nma:output>
</nma:rpc>
```

As indicated in the schema fragment, contents of the 'input' substatement (if any) are mapped under `<rng:element name="PREFIX:ARGUMENT">`. Similarly, contents of the 'output' substatement are mapped under `<nma:output">`. If there is no 'output' substatement, the `<nma:output>` element MUST NOT be present.

The `<nma:rpc>` element is a child of `<nma:rpcs>`.

10.51. The 'status' Statement

This statement MAY be ignored. Otherwise, it is mapped to `@nma:status` attribute and `ARGUMENT` becomes its value.

10.52. The 'submodule' Statement

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

10.53. The 'type' Statement

Most YANG built-in datatypes have an equivalent in the XSD datatype library [XSD-D] as shown in Table 4.

| 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 |
| string | string | character string |
| binary | base64Binary | binary data in base64 encoding |

Table 4: YANG built-in datatypes with equivalents in the W3C XML Schema Type Library

Two important datatypes of the XSD datatype library - "dateTime" and "anyURI" - are not built-in types in YANG but instead are defined as derived types in the standard modules [RFC6021]: "date-and-time" in the "ietf-yang-types" module and "uri" in the "ietf-inet-types" module. However, the formal restrictions in the YANG type definitions are rather weak. Therefore, implementations of the YANG-to-DSDL mapping SHOULD detect these derived types in source YANG modules and map them to "dateTime" and "anyURI", respectively.

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

10.53.1. The "empty" Type

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

10.53.2. The "boolean" Type

This built-in type does not allow any restrictions and is mapped to the following XML fragment:

```
<rng:choice>
  <rng:value>true</rng:value>
  <rng:value>false</rng:value>
</rng:choice>
```

Note that the XSD "boolean" type cannot be used here because it allows, unlike YANG, an alternative numeric representation of boolean values: 0 for "false" and 1 for "true".

10.53.3. The "binary" Type

This built-in type does not allow any restrictions and is mapped simply by inserting `<rng:data>` element whose `@type` attribute value is set to "base64Binary" (see also Table 4).

10.53.4. The "bits" Type

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 a 'bit' substatement.

10.53.5. The "enumeration" and "union" Types

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

10.53.6. The "identityref" Type

This type is mapped to the following named pattern reference:

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

where `PREFIX:BASE` is the qualified name of the identity appearing in the argument of the 'base' substatement.

For example, assume that module "des" in Section 10.21 contains the following leaf definition:

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

This leaf would then be mapped to the following element pattern:

```
<element name="des:foo">
  <ref name="__crypto_crypto-alg"/>
</element>
```

10.53.7. The "instance-identifier" Type

This type is mapped to `<rng:data>` element with `@type` attribute set to "string". In addition, an empty `<nma:instance-identifier>` element MUST be inserted as a child of PARENT.

The argument of the 'require-instance' substatement, if it exists, becomes the value of the `@require-instance` attribute of the `<nma:instance-identifier>` element.

10.53.8. The "leafref" Type

This type is mapped exactly as the type of the leaf given in the argument of 'path' substatement. However, if the type of the referred leaf defines a default value, this default value MUST be ignored by the mapping.

In addition, `@nma:leafref` attribute MUST be added to PARENT. The argument of the 'path' substatement, translated according to Section 9.3, is set as the value of this attribute.

10.53.9. The Numeric Types

YANG built-in numeric types are "int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64" and "decimal64". They are mapped to `<rng:data>` element with `@type` attribute set to ARGUMENT translated according to Table 4 above.

An exception is the "decimal64" type, which is mapped to the "decimal" type of the XSD datatype library. Its precision and number of fractional digits are controlled with the following facets, which MUST always be present:

- o "totalDigits" facet set to the value of 19.
- o "fractionDigits" facet set to the argument of the 'fraction-digits' substatement.

The fixed value of "totalDigits" corresponds to the maximum of 19 decimal digits for 64-bit integers.

For example, the statement

```
type decimal64 {  
    fraction-digits 2;  
}
```

is mapped to the following RELAX NG datatype:

```
<rng:data type="decimal">  
    <rng:param name="totalDigits">19</rng:param>  
    <rng:param name="fractionDigits">2</rng:param>  
</rng:data>
```

All numeric types support the 'range' restriction, which is mapped as follows:

If the range expression consists of just a single range LO..HI, then it is mapped to a pair of datatype facets

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

and

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

If the range consists of a single number, the values of both facets are set to this value. If LO is equal to the string "min", the "minInclusive" facet is omitted. If HI is equal to the string "max", the "maxInclusive" facet is omitted.

If the range expression has multiple parts separated by "|", then the parent <rng:data> element must be repeated once for every range part and all such <rng:data> elements are wrapped 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 paragraph.

For the "decimal64" type, the "totalDigits" and "fractionDigits" must be repeated inside each of the <rng:data> elements.

For example,

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

is mapped to the following RELAX NG fragment:

```

<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>

```

See Section 9.2.2 for further details on mapping the restrictions.

10.53.10. The "string" Type

This type is mapped to `<rng:data>` element with the `@type` attribute set to "string".

The 'length' restriction is handled analogically to the 'range' restriction for the numeric types (Section 10.53.9):

If the length expression has just a single range, then

- o if the length range consists of a single number LENGTH, the following datatype facet is inserted:

```
<rng:param name="length">LENGTH</rng:param>.
```

- o Otherwise the length range is of the form LO..HI, i.e., it consists of both the lower and upper bound. The following two datatype facets are then inserted:

```
<rng:param name="minLength">LO</rng:param>
```

and

```
<rng:param name="maxLength">HI</rng:param>
```

If LO is equal to the string "min", the "minLength" facet is omitted. If HI is equal to the string "max", the "maxLength" facet is omitted.

If the length expression has of multiple parts separated by "|", then the parent `<rng:data>` element must be repeated once for every range part and all such `<rng:data>` elements are wrapped in `<rng:choice>`

element. Each `<rng:data>` element contains the "length" or "minLength" and "maxLength" facets for one part of the length expression as described in the previous paragraph.

Every 'pattern' restriction of the "string" datatype is mapped to the "pattern" facet

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

with text equal to the argument of the 'pattern' statement. All such "pattern" facets must be repeated inside each copy of the `<rng:data>` element, i.e., once for each length range.

For example,

```
type string {  
    length "1|3..8";  
    pattern "[A-Z][a-z]*";  
}
```

is mapped to the following RELAX NG fragment:

```
<rng:choice>  
  <rng:data type="string">  
    <rng:param name="length">1</rng:param>  
    <rng:param name="pattern">[A-Z][a-z]*</rng:param>  
  </rng:data>  
  <rng:data type="string">  
    <rng:param name="minLength">3</rng:param>  
    <rng:param name="maxLength">8</rng:param>  
    <rng:param name="pattern">[A-Z][a-z]*</rng:param>  
  </rng:data>  
</rng:choice>
```

10.53.11. Derived Types

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 hybrid schema yet, the corresponding type definition **MUST** be found and its mapping installed as a subelement of either the root or an embedded `<rng:grammar>` element, see Section 10.54. 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 ancestor built-in type for the given derived type must be determined and the mapping of this base type MUST be used. Restrictions appearing at all stages of the type derivation chain MUST be taken into account and their conjunction added to the `<rng:data>` element which defines the basic type.

See Section 9.2.2 for more details and an example.

10.54. The 'typedef' Statement

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 either the root or an embedded `<rng:grammar>` element, depending on whether the 'typedef' statement appears at the top level of a YANG module or not. The name of this named pattern definition is set to ARGUMENT mangled according to the rules specified in Section 9.2.

Whenever a derived type is used with additional restrictions, the ancestor built-in 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 10.53.11 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.

10.55. The 'unique' Statement

This statement is mapped to `@nma:unique` attribute. ARGUMENT MUST be 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"
```

10.56. The 'units' Statement

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

10.57. The 'uses' Statement

If this statement has neither 'refine' nor 'augment' substatements, it is mapped to <rng:ref> element, i.e., a reference to a named pattern, and the value of its @name attribute is set to ARGUMENT mangled according to Section 9.2. If the RELAX NG definition of the referenced named pattern has not been added to the hybrid schema yet, the corresponding grouping MUST be found and its mapping installed as a subelement of <rng:grammar>, see Section 10.20.

Otherwise, if the 'uses' statement has any 'refine' or 'augment' substatements, the corresponding grouping must be looked up and its contents inserted under PARENT. See Section 9.2.1 for further details and an example.

10.58. The 'value' Statement

This statement is ignored.

10.59. The 'when' Statement

This statement is mapped to @nma:when attribute and ARGUMENT, translated according to Section 9.3, becomes its value.

10.60. The 'yang-version' Statement

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). In the case of a mismatch, the implementation SHOULD report an error and terminate.

10.61. The 'yin-element' Statement

This statement is not mapped to the output schema, but see the rules for extension handling in Section 9.4.

11. Mapping the Hybrid Schema to DSDL

As explained in Section 6, the second step of the YANG-to-DSDL mapping takes the hybrid schema and transforms it to various DSDL schemas capable of validating instance XML documents. As an input parameter, this step takes, in the simplest case, just a specification of the NETCONF XML document type that is to be validated. These document types can be, for example, the contents of a datastore, a reply to `<nc:get>` or `<nc:get-config>`, contents of other RPC requests/replies and event notifications, and so on.

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

1. Extract the parts of the hybrid schema that are appropriate for the requested document type. For example, if a `<nc:get>` reply is to be validated, the subtree under `<nma:data>` has to be selected.
2. The schema must be adapted 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 `<nc:get>` reply 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 the corresponding patterns or rules.

These three tasks are together much simpler than the first mapping step and can be effectively implemented using XSL transformations [XSLT].

The following subsections describe the details of the second mapping step for the individual DSDL schema languages. Section 12 then contains a detailed specification for the mapping of all NETMOD-specific annotations.

11.1. Generating RELAX NG Schemas for Various Document Types

With one minor exception, obtaining a validating RELAX NG schema from the hybrid schema only means taking appropriate parts of the hybrid schema and assembling them in a new RELAX NG grammar, perhaps after removing all unwanted annotations.

The structure of the resulting RELAX NG schema is similar to that of the hybrid schema: The root grammar contains embedded grammars, one for each input YANG module. However, as explained in Section 8.2, global named pattern definitions (children of the root `<rng:grammar>` element) MUST be moved to a separate schema file.

Depending on the XML document type that is the target for validation, such as <nc:get>/<nc:get-config> reply, RPC operations or notifications, patterns defining corresponding top-level information items MUST be added, such as <nc:rpc-reply> with the @message-id attribute and so on.

In order to avoid copying common named pattern definitions for common NETCONF elements and attributes to every single output RELAX NG file, such schema-independent definitions SHOULD be collected in a library file which is then included by the validating RELAX NG schemas. Appendix B has the listing of such a library file.

The minor exception mentioned above is the annotation @nma:config, which must be observed if the target document type is a reply to <nc:get-config>. In this case, each element definition that has this attribute with the value of "false" MUST be removed from the schema together with its descendants. See Section 12.1 for more details.

11.2. Mapping Semantic Constraints to Schematron

Schematron schemas tend to be much flatter and more uniform compared to RELAX NG. They have 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. The following NETMOD-specific annotations from the hybrid schema (henceforth called "semantic annotations") are mapped to corresponding Schematron rules: <nma:must>, @nma:key, @nma:unique, @nma:max-elements, @nma:min-elements, @nma:when, @nma:leafref, @nma:leaf-list, and also @nma:mandatory appearing as an attribute of <rng:choice> (see Section 11.2.1).

Each input YANG module is mapped to a Schematron pattern whose @id attribute is set to the module name. Every <rng:element> pattern containing at least one of the above-mentioned semantic annotations is then mapped to a Schematron rule:

```
<sch:rule context="XELEM">
  ...
</sch:rule>
```

The value of the mandatory @context attribute of <sch:rule> (denoted as XELEM) MUST be set to the absolute path of the context element in the data tree. The <sch:rule> element contains the mappings of all contained semantic annotations in the form of Schematron asserts or reports.

Semantic annotations appearing inside a named pattern definition (i.e., having `<rng:define>` among its ancestors) require special treatment because they may be potentially used in different contexts. This is accomplished by using Schematron abstract patterns that use the `"$pref"` variable in place of the local namespace prefix. The value of the `@id` attribute of such an abstract pattern MUST be set to the name of the named pattern definition which is being mapped (i.e., the mangled name of the original YANG grouping).

When the abstract pattern is instantiated, the values of the following two parameters MUST be provided:

- o `pref`: the actual namespace prefix,
- o `start`: XPath expression defining the context in which the grouping is used.

EXAMPLE. Consider the following YANG module:

```
module example4 {
  namespace "http://example.com/ns/example4";
  prefix ex4;
  uses sorted-leaf-list;
  grouping sorted-leaf-list {
    leaf-list sorted-entry {
      must "not(preceding-sibling::sorted-entry > .)" {
        error-message "Entries must appear in ascending order.";
      }
      type uint8;
    }
  }
}
```

The resulting Schematron schema for a reply to `<nc:get>` is then as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:ns uri="http://example.com/ns/example4" prefix="ex4"/>
  <sch:ns uri="urn:ietf:params:xml:ns:netconf:base:1.0"
    prefix="nc"/>
  <sch:pattern abstract="true"
    id="_example4__sorted-leaf-list">
    <sch:rule context="$start/$pref:sorted-entry">
      <sch:report
        test=". = preceding-sibling::$pref:sorted-entry">
        Duplicate leaf-list entry "<sch:value-of select="."/>".
      </sch:report>
      <sch:assert
        test="not(preceding-sibling::$pref:sorted-entry > .)">
        Entries must appear in ascending order.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:pattern id="example4"/>
  <sch:pattern id="id2573371" is-a="_example4__sorted-leaf-list">
    <sch:param name="start" value="/nc:rpc-reply/nc:data"/>
    <sch:param name="pref" value="ex4"/>
  </sch:pattern>
</sch:schema>

```

The "sorted-leaf-list" grouping from the input module is mapped to an abstract pattern with an @id value of "_example4__sorted-leaf-list" in which the 'must' statement corresponds to the <sch:assert> element. The abstract pattern is instantiated by the pattern with an @id value of "id2802112" which sets the "start" and "pref" parameters to appropriate values.

Note that another Schematron element, <sch:report>, was automatically added, checking for duplicate leaf-list entries.

The mapping from the hybrid schema to Schematron proceeds in the following steps:

1. First, the active subtree(s) of the hybrid 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 <nc: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/example4" prefix="ex4"/>
```

3. One pattern is created for every input module. In addition, an abstract pattern is created for every named pattern definition containing one or more semantic annotations.
4. A <sch:rule> element is created for each element pattern containing semantic annotations.
5. Every such annotation is then mapped to an <sch:assert> or <sch:report> element which is installed as a child of the <sch:rule> element.

11.2.1. Constraints on Mandatory Choice

In order to fully represent the semantics of YANG's 'choice' statement with the "mandatory true;" substatement, the RELAX NG grammar has to be combined with a special Schematron rule.

EXAMPLE. Consider the following module:

```
module example5 {  
  namespace "http://example.com/ns/example5";  
  prefix ex5;  
  choice foobar {  
    mandatory true;  
    case foo {  
      leaf foo1 {  
        type uint8;  
      }  
      leaf foo2 {  
        type uint8;  
      }  
    }  
    leaf bar {  
      type uint8;  
    }  
  }  
}
```

In this module, all three leaf nodes in both case branches are optional but because of the "mandatory true;" statement, at least one of them must be present in a valid configuration. The 'choice' statement from this module is mapped to the following fragment of the RELAX NG schema:

```
<rng:choice>
  <rng:interleave>
    <rng:optional>
      <rng:element name="ex5:foo1">
        <rng:data type="unsignedByte"/>
      </rng:element>
    </rng:optional>
    <rng:optional>
      <rng:element name="ex5:foo2">
        <rng:data type="unsignedByte"/>
      </rng:element>
    </rng:optional>
  </rng:interleave>
  <rng:element name="ex5:bar">
    <rng:data type="unsignedByte"/>
  </rng:element>
</rng:choice>
```

In the second case branch, the "ex5:bar" element is defined as mandatory so that this element must be present in a valid configuration if this branch is selected. However, the two elements in the first branch "foo" cannot be both declared as mandatory since each of them alone suffices for a valid configuration. As a result, the above RELAX NG fragment would successfully validate configurations where none of the three leaf elements are present.

Therefore, mandatory choices, which can be recognized in the hybrid schema as <rng:choice> elements with the @nma:mandatory annotation, have to be handled in a special way: For each mandatory choice where at least one of the cases contains more than one node, a Schematron rule MUST be added enforcing the presence of at least one element from any of the cases. (RELAX NG schema guarantees that elements from different cases cannot be mixed together, that all mandatory nodes are present etc.).

For the example module above, the Schematron rule will be as follows:

```
<sch:rule context="/nc:rpc-reply/nc:data">
  <sch:assert test="ex5:foo1 or ex5:foo2 or ex5:bar">
    Node(s) from at least one case of choice "foobar" must exist.
  </sch:assert>
</sch:rule>
```

11.3. Mapping Default Values to DSRL

DSRL is the only component of DSDL which is allowed to change the information set of the validated XML document. While DSRL also has other functions, YANG-to-DSDL mapping uses it only for specifying and

applying default contents. For XML instance documents based on YANG data models, insertion of default contents may potentially take place for all implicit nodes identified by the rules in Section 9.1.2.

In DSRL, the default contents of an element are specified using the `<dsrl:default-content>` element, which is a child of `<dsrl:element-map>`. Two sibling elements of `<dsrl:default-content>` determine the context for the application of the default contents, see [DSRL]:

- o `<dsrl:parent>` element contains an XSLT pattern specifying the parent element; the default contents are applied only if the parent element exists in the instance document.
- o `<dsrl:name>` contains the XML name of the element which, if missing or empty, is inserted together with the contents of `<dsrl:default-content>`.

The `<dsrl:parent>` element is optional in a general DSRL schema but, for the purpose of the YANG-to-DSDL mapping, this element **MUST** be always present, in order to guarantee a proper application of default contents.

DSRL mapping only deals with `<rng:element>` patterns in the hybrid schema that define implicit nodes (see Section 9.1.2). Such element patterns are distinguished by having NETMOD-specific annotation attributes `@nma:default` or `@nma:implicit`, i.e., either

```
<rng:element name="ELEM" nma:default="DEFVALUE">
  ...
</rng:element>
```

or

```
<rng:element name="ELEM" nma:implicit="true">
  ...
</rng:element>
```

The former case applies to leaf nodes having the 'default' substatement, but also to leaf nodes that obtain their default value from a typedef, if this typedef is expanded according to the rules in Section 9.2.2 so that the `@nma:default` annotation is attached directly to the leaf's element pattern.

The latter case is used for all implicit containers (see Section 9.1) and for leaves that obtain the default value from a typedef and don't have the `@nma:default` annotation.

In the simplest case, both element patterns are mapped to the

following DSRL element map:

```
<dsrl:element-map>
  <dsrl:parent>XPARENT</dsrl:parent>
  <dsrl:name>ELEM</dsrl:name>
  <dsrl:default-content>DEFCONT</dsrl:default-content>
</dsrl:element-map>
```

where XPARENT is the absolute XPath of ELEM's parent element in the data tree and DEFCONT is constructed as follows:

- o If the implicit node ELEM is a leaf and has the @nma:default attribute, DEFCONT is set to the value of this attribute (denoted above as DEFVALUE).
- o If the implicit node ELEM is a leaf and has the @nma:implicit attribute with the value of "true", the default value has to be determined from the @nma:default attribute of the definition of ELEM's type (perhaps recursively) and used in place of DEFCONT in the above DSRL element map. See also Section 9.2.2.
- o Otherwise, the implicit node ELEM is a container and DEFCONT is constructed as an XML fragment containing all descendant elements of ELEM that have either @nma:implicit or @nma:default attribute.

In addition, when mapping the default case of a choice, it has to be guaranteed that the default contents are not applied if any node from any non-default case is present. This is accomplished by setting <dsrl:parent> in a special way:

```
<dsrl:parent>XPARENT[not (ELEM1|ELEM2|...|ELEMn)]</dsrl:parent>
```

where ELEM1, ELEM2, ... ELEMn are the names of all top-level nodes from all non-default cases. The rest of the element map is exactly as before.

EXAMPLE. Consider the following YANG module:


```
module example6 {  
  namespace "http://example.com/ns/example6";  
  prefix ex6;  
  container outer {  
    leaf leaf1 {  
      type uint8;  
      default 1;  
    }  
    choice one-or-two {  
      default "one";  
      container one {  
        leaf leaf2 {  
          type uint8;  
          default 2;  
        }  
      }  
      leaf leaf3 {  
        type uint8;  
        default 3;  
      }  
    }  
  }  
}
```

The DSRL schema generated for the "get-reply" target document type will be:

```
<?xml version="1.0" encoding="utf-8"?>
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
           xmlns:ex6="http://example.com/ns/example6"
           xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data</dsrl:parent>
    <dsrl:name>ex6:outer</dsrl:name>
    <dsrl:default-content>
      <ex6:leaf1>1</ex6:leaf1>
      <ex6:one>
        <ex6:leaf2>2</ex6:leaf2>
      </ex6:one>
    </dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data/ex6:outer</dsrl:parent>
    <dsrl:name>ex6:leaf1</dsrl:name>
    <dsrl:default-content>1</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/ex6:outer[not(ex6:leaf3)]
    </dsrl:parent>
    <dsrl:name>ex6:one</dsrl:name>
    <dsrl:default-content>
      <ex6:leaf2>2</ex6:leaf2>
    </dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/ex6:outer/ex6:one
    </dsrl:parent>
    <dsrl:name>ex6:leaf2</dsrl:name>
    <dsrl:default-content>2</dsrl:default-content>
  </dsrl:element-map>
</dsrl:maps>
```

Note that the default value for "leaf3" defined in the YANG module is ignored because "leaf3" represents a non-default alternative of a choice and as such never becomes an implicit element.

12. Mapping NETMOD-specific Annotations to DSDL Schema Languages

This section contains the mapping specification for the 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 11. The context is determined by the element pattern in the hybrid schema to which the annotation is attached. In the rest of this section, CONTELEM will denote the name of this context element properly qualified with its namespace prefix.

12.1. The @nma:config Annotation

If this annotation is present with the value of "false", the following rules MUST be observed for DSDL schemas of <nc:get-config> reply:

- o When generating RELAX NG, the contents of the CONTELEM definition MUST be changed to <rng:notAllowed>.
- o When generating Schematron or DSRL, the CONTELEM definition and all its descendants in the hybrid schema MUST be ignored.

12.2. The @nma:default Annotation

This annotation is used for generating the DSRL schema as described in Section 11.3.

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

This annotation currently has no mapping defined.

12.4. The <nma:error-message> Annotation

This annotation is handled within <nma:must>, see Section 12.13.

12.5. The @if-feature Annotation

The information about available features MAY be supplied as an input parameter to an implementation. In this case, the following changes MUST be performed for all features that are considered unavailable:

- o When generating RELAX NG, the contents of the CONTELEM definition MUST be changed to <rng:notAllowed>.
- o When generating Schematron or DSRL, the CONTELEM definition and all its descendants in the hybrid schema MUST be ignored.

12.6. The @nma:implicit Annotation

This annotation is used for generating the DSRL schema as described in Section 11.3.

12.7. The <nma:instance-identifier> Annotation

If this annotation element has the @require-instance attribute with the value of "false", it is ignored. Otherwise it is mapped to the following Schematron assert:

```
<sch:assert test="nmf:evaluate(.)">
  The element pointed to by "CONTELEM" must exist.
</sch:assert>
```

The nmf:evaluate() function is an XSLT extension function (see Extension Functions in [XSLT]) that evaluates an XPath expression at run time. Such an extension function is available in Extended XSLT (EXSLT) or provided as a proprietary extension by some XSLT processors, for example Saxon.

12.8. The @nma:key Annotation

Assume this annotation attribute contains "k_1 k_2 ... k_n", i.e., specifies n children of CONTELEM as list 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 sub-expression C_i, for i=1,2,...,n, specifies the condition for violated uniqueness of the key k_i, namely

k_i=current()/k_i

12.9. The @nma:leaf-list Annotation

This annotation is mapped to the following Schematron rule which detects duplicate entries of a leaf-list:

```
<sch:report
  test=". = preceding-sibling::PREFIX:sorted-entry">
  Duplicate leaf-list entry "<sch:value-of select="."/>".
</sch:report>
```

See Section 11.2 for a complete example.

12.10. The @nma:leafref Annotation

This annotation is mapped to the following assert:

```
<sch:assert test="PATH=.">  
  Leaf "PATH" does not exist for leafref value "VALUE"  
</sch:assert>
```

where PATH is the value of @nma:leafref and VALUE is the value of the context element in the instance document for which the referred leaf doesn't exist.

12.11. The @nma:min-elements Annotation

This annotation is mapped to the following Schematron assert:

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

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

12.12. The @nma:max-elements Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert  
  test="count(../CONTELEM)<=MAX or preceding-sibling:../CONTELEM">  
  Number of list items must be at most MAX  
</sch:assert>
```

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

12.13. The <nma:must> Annotation

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 contents, otherwise it is set to the default message "Condition EXPRESSION must be true".

12.14. The <nma:ordered-by> Annotation

This annotation currently has no mapping defined.

12.15. The <nma:status> Annotation

This annotation currently has no mapping defined.

12.16. The @nma:unique Annotation

The mapping of this annotation is almost identical as for @nma:key, see Section 12.8, with two small differences:

- o 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 12.8 work without any modifications if the descendant schema node identifiers are substituted for k_1, k_2, ..., k_n.
- o The message appearing as the text of <sch:report> is different: "Violated uniqueness for list CONTELEM".

12.17. The @nma:when Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert test="EXPRESSION">  
  Node "CONTELEM" is only valid when "EXPRESSION" is true.  
</sch:assert>
```

where EXPRESSION is the value of @nma:when.

13. IANA Considerations

This document requests the following two registrations of namespace URIs in the IETF XML registry [RFC3688]:

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

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:netmod:xpath-extensions:1

Registrant Contact: The IESG.

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

14. Security Considerations

This document defines a procedure that maps data models expressed in the YANG language to a coordinated set of DSDL schemas. The procedure itself has no security impact on the Internet.

DSDL schemas obtained by the mapping procedure may be used for validating the contents of NETCONF messages or entire datastores and thus provide additional validity checks above those performed by NETCONF server and client implementations supporting YANG data models. The strictness of this validation is directly derived from the source YANG modules that the validated XML data adhere to.

15. Contributors

The following people contributed significantly to the initial version of this document:

- o Rohan Mahy
- o Sharon Chisholm (Ciena)

16. Acknowledgments

The editor wishes to thank the following individuals who provided helpful suggestions and/or comments on various versions of this document: Andy Bierman, Martin Bjorklund, Jirka Kosek, Juergen Schoenwaelder and Phil Shafer.

17. References

17.1. Normative References

- [DSDL] ISO/IEC, "Document Schema Definition Languages (DSDL) - Part 1: Overview", ISO/IEC 19757-1, November 2004.
- [DSRL] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 8: Document Semantics Renaming Language - DSRL", ISO/IEC 19757-8:2008(E), December 2008.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, September 2010.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, September 2010.
- [RNG] 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), December 2008.
- [RNG-CS] 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), January 2006.
- [RNG-DTD] Clark, J., Ed. and M. Murata, Ed., "RELAX NG DTD Compatibility", OASIS Committee Specification 3 December 2001, December 2001.
- [Schematron] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron", ISO/IEC 19757-3:2006(E), June 2006.

- [XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2006/REC-xml-20060816>>.
- [XML-INFOSET] Tobin, R. and J. Cowan, "XML Information Set (Second Edition)", World Wide Web Consortium Recommendation REC-xml-infoset-20040204, February 2004, <<http://www.w3.org/TR/2004/REC-xml-infoset-20040204>>.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.
- [XSD-D] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.
- [XSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999.

17.2. Informative References

- [DHCPTut] Bjorklund, M., "DHCP Tutorial", November 2007, <<http://www.yang-central.org/twiki/bin/view/Main/DhcpTutorial>>.
- [RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", STD 15, RFC 1157, May 1990.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.
- [RFC5013] Kunze, J., "The Dublin Core Metadata Element Set", RFC 5013, August 2007.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [Trang] Clark, J., "Trang: Multiformat schema converter based on RELAX NG", 2008, <<http://www.thaiopensource.com/relaxng/trang.html>>.

- [Vli04] van der Vlist, E., "RELAX NG", O'Reilly , 2004.
- [XSD] 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, <<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>>.
- [pyang] Bjorklund, M. and L. Lhotka, "pyang: An extensible YANG validator and converter in Python", 2010, <<http://code.google.com/p/pyang/>>.

Appendix A. RELAX NG Schema for NETMOD-Specific Annotations

This appendix defines the content model for all NETMOD-specific annotations in the form of RELAX NG named pattern definitions.

```
<CODE BEGINS> file "nmannot.rng"

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

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

  <define name="data-element">
    <element name="nma:data">
      <ref name="__anyxml__"/>
    </element>
  </define>

  <define name="default-attribute">
    <attribute name="nma:default">
      <data type="string"/>
    </attribute>
  </define>

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

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

  <define name="if-feature-attribute">
    <attribute name="nma:if-feature">
      <list>
        <data type="QName"/>
      </list>
    </attribute>
  </define>
```

```
</define>

<define name="implicit-attribute">
  <attribute name="nma:implicit">
    <data type="boolean"/>
  </attribute>
</define>

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

<define name="key-attribute">
  <attribute name="nma:key">
    <list>
      <data type="QName"/>
    </list>
  </attribute>
</define>

<define name="leaf-list-attribute">
  <attribute name="nma:leaf-list">
    <data type="boolean"/>
  </attribute>
</define>

<define name="leafref-attribute">
  <attribute name="nma:leafref">
    <data type="string"/>
  </attribute>
</define>

<define name="mandatory-attribute">
  <attribute name="nma:mandatory">
    <data type="Name"/>
  </attribute>
</define>

<define name="max-elements-attribute">
  <attribute name="nma:max-elements">
    <data type="nonNegativeInteger"/>
  </attribute>
```

```
</define>

<define name="min-elements-attribute">
  <attribute name="nma:min-elements">
    <data type="nonNegativeInteger"/>
  </attribute>
</define>

<define name="module-attribute">
  <attribute name="nma:module">
    <data type="Name"/>
  </attribute>
</define>

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

<define name="notifications-element">
  <element name="nma:notifications">
    <zeroOrMore>
      <element name="nma:notification">
        <ref name="__anyxml__"/>
      </element>
    </zeroOrMore>
  </element>
</define>

<define name="rpcs-element">
  <element name="nma:rpcs">
    <zeroOrMore>
      <element name="nma:rpc">
        <interleave>
          <element name="nma:input">
            <ref name="__anyxml__"/>
          </element>
        </interleave>
      </element>
    </zeroOrMore>
  </element>
</define>
```



```
        <optional>
          <element name="nma:output">
            <ref name="__anyxml__"/>
          </element>
        </optional>
      </interleave>
    </element>
  </zeroOrMore>
</element>
</define>
```

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

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

```
<define name="unique-attribute">
  <optional>
    <attribute name="nma:unique">
      <list>
        <data type="token"/>
      </list>
    </attribute>
  </optional>
</define>
```

```
<define name="units-attribute">
  <optional>
    <attribute name="nma:units">
      <data type="string"/>
    </attribute>
  </optional>
</define>
```

```
</define>

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

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

</grammar>

<CODE ENDS>
```

Appendix B. Schema-Independent Library

In order to avoid copying the common named pattern definitions to every RELAX NG schema generated in the second mapping step, the definitions are collected in 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 [RFC4741] and event notifications [RFC5277].

```
<CODE BEGINS> file "relaxng-lib.rng"

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

<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>

<CODE ENDS>
```

Appendix C. Mapping DHCP Data Model - A Complete Example

This appendix demonstrates both steps of the YANG-to-DSDL mapping applied to the "canonical" DHCP tutorial [DHCPtut] data model. The input YANG module is shown in Appendix C.1 and the output schemas in the following two subsections.

The hybrid schema was obtained by the "dsdl" plugin of the pyang tool [pyang] and the validating DSDL schemas were obtained by XSLT stylesheets that are also part of pyang distribution.

Due to the limit of 72 characters per line, a few long strings required manual editing, in particular the regular expression patterns for IP addresses etc. These were 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.

C.1. Input YANG Module

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

  import ietf-yang-types { prefix yang; }
  import ietf-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 '.. <= ../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. Hybrid Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:nma="urn:ietf:params:xml:ns:netmod:dSDL-annotations:1"
  xmlns:dc="http://purl.org/dc/terms"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dhcp="http://example.com/ns/dhcp"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <dc:creator>Pyang 1.0a, DSDL plugin</dc:creator>
  <dc:date>2010-06-17</dc:date>
  <start>
    <grammar nma:module="dhcp" ns="http://example.com/ns/dhcp">
      <dc:source>YANG module 'dhcp'</dc:source>
      <start>
        <nma:data>
          <optional>
            <element nma:implicit="true" name="dhcp:dhcp">
              <interleave>
                <a:documentation>
                  configuration and operational parameters for a DHCP server.
                </a:documentation>
              </optional>
              <element nma:default="7200"
                name="dhcp:max-lease-time"
                nma:units="seconds">
                <data type="unsignedInt"/>
              </element>
            </optional>
            <optional>
              <element nma:default="600"
                name="dhcp:default-lease-time"
                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 nma:key="dhcp:name"
                    name="dhcp:shared-network">
                    <element name="dhcp:name">
                      <data type="string"/>
                    </element>
                  </element>
                </zeroOrMore>
              </element>
            </optional>
          </optional>
        </nma:data>
      </start>
    </grammar>
  </start>
</grammar>
```

```
        </element>
        <ref name="_dhcp__subnet-list"/>
    </element>
</zeroOrMore>
</element>
</optional>
<optional>
    <element name="dhcp:status" nma:config="false">
        <zeroOrMore>
            <element nma:key="dhcp:address"
                name="dhcp:leases">
                <element name="dhcp:address">
                    <ref name="ietf-inet-types__ip-address"/>
                </element>
                <interleave>
                    <optional>
                        <element name="dhcp:starts">
                            <ref name="ietf-yang-types__date-and-time"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="dhcp:ends">
                            <ref name="ietf-yang-types__date-and-time"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="dhcp:hardware">
                            <interleave>
                                <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="ietf-yang-types__phys-address"/>
                                    </element>
                                </optional>
                            </interleave>
                        </element>
                    </optional>
                </interleave>
            </element>
        </zeroOrMore>
    </optional>
</optional>
```



```
        </element>
      </optional>
    </interleave>
  </element>
</optional>
</nma:data>
<nma:rpcs/>
<nma:notifications/>
</start>
</grammar>
</start>
<define name="ietf-yang-types__phys-address">
  <data type="string">
    <param name="pattern">([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?</param>
  </data>
</define>
<define name="ietf-inet-types__ipv6-address">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="ietf-inet-types__ip-prefix">
  <choice>
    <ref name="ietf-inet-types__ipv4-prefix"/>
    <ref name="ietf-inet-types__ipv6-prefix"/>
  </choice>
</define>
<define name="ietf-inet-types__host">
  <choice>
    <ref name="ietf-inet-types__ip-address"/>
    <ref name="ietf-inet-types__domain-name"/>
  </choice>
</define>
<define name="ietf-yang-types__date-and-time">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="_dhcp__subnet-list">
  <a:documentation>A reusable list of subnets</a:documentation>
  <zeroOrMore>
    <element nma:key="net" name="subnet">
      <element name="net">
        <ref name="ietf-inet-types__ip-prefix"/>
      </element>
    </element>
  </interleave>
  <optional>
```

```
<element name="range">
  <interleave>
    <optional>
      <element name="dynamic-bootp">
        <a:documentation>
          Allows BOOTP clients to get addresses in this range
        </a:documentation>
        <empty/>
      </element>
    </optional>
    <element name="low">
      <ref name="ietf-inet-types__ip-address"/>
    </element>
    <element name="high">
      <ref name="ietf-inet-types__ip-address"/>
    </element>
  </interleave>
</element>
</optional>
<optional>
  <element name="dhcp-options">
    <interleave>
      <a:documentation>
        Options in the DHCP protocol
      </a:documentation>
      <zeroOrMore>
        <element nma:leaf-list="true" name="router"
          nma:ordered-by="user">
          <a:documentation>
            See: RFC 2132, sec. 3.8
          </a:documentation>
          <ref name="ietf-inet-types__host"/>
        </element>
      </zeroOrMore>
      <optional>
        <element name="domain-name">
          <a:documentation>
            See: RFC 2132, sec. 3.17
          </a:documentation>
          <ref name="ietf-inet-types__domain-name"/>
        </element>
      </optional>
    </interleave>
  </element>
</optional>
<optional>
  <element nma:default="7200" name="max-lease-time"
    nma:units="seconds">
```

```
        <data type="unsignedInt"/>
      </element>
    </optional>
  </interleave>
</element>
</zeroOrMore>
</define>
<define name="ietf-inet-types__domain-name">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
    <param name="minLength">1</param>
    <param name="maxLength">253</param>
  </data>
</define>
<define name="ietf-inet-types__ipv4-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="ietf-inet-types__ipv4-address">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="ietf-inet-types__ipv6-prefix">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
    <param name="pattern">... regex pattern ...</param>
  </data>
</define>
<define name="ietf-inet-types__ip-address">
  <choice>
    <ref name="ietf-inet-types__ipv4-address"/>
    <ref name="ietf-inet-types__ipv6-address"/>
  </choice>
</define>
</grammar>
```

C.3. Final DSDL Schemas

This appendix contains DSDL schemas that were obtained from the hybrid schema in Appendix C.2 by XSL transformations. These schemas can be directly used for validating a reply to unfiltered <nc:get> with the contents corresponding to the DHCP data model.

The RELAX NG schema (in two parts, as explained in Section 8.2) also includes the schema-independent library from Appendix B.

C.3.1. Main RELAX NG Schema for <nc:get> Reply

```
<?xml version="1.0" encoding="utf-8"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:nma="urn:ietf:params:xml:ns:netmod:dSDL-annotations:1"
  xmlns:dhcp="http://example.com/ns/dhcp"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
  ns="urn:ietf:params:xml:ns:netconf:base:1.0">
<include href="relaxng-lib.rng"/>
<start>
  <element name="rpc-reply">
    <ref name="message-id-attribute"/>
    <element name="data">
      <interleave>
        <grammar ns="http://example.com/ns/dhcp">
          <include href="dhcp-gdefs.rng"/>
          <start>
            <optional>
              <element name="dhcp:dhcp">
                <interleave>
                  <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="ietf-inet-types__ip-address"/>
</element>
<interleave>
  <optional>
    <element name="dhcp:starts">
      <ref name="ietf-yang-types__date-and-time"/>
    </element>
  </optional>
  <optional>
    <element name="dhcp:ends">
      <ref name="ietf-yang-types__date-and-time"/>
    </element>
  </optional>
  <optional>
    <element name="dhcp:hardware">
      <interleave>
        <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="ietf-yang-types__phys-address"/>
          </element>
        </optional>
      </interleave>
    </element>
  </optional>
</interleave>
</element>
</zeroOrMore>
</element>
</optional>
</interleave>
</element>
</optional>
</start>
</grammar>
</interleave>
</element>
</element>
</start>
```

</grammar>

C.3.2. RELAX NG Schema - Global Named Pattern Definitions

```
<?xml version="1.0" encoding="utf-8"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:nma="urn:ietf:params:xml:ns:netmod:dSDL-annotations:1"
  xmlns:dhcp="http://example.com/ns/dhcp"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <define name="ietf-yang-types__phys-address">
    <data type="string">
      <param name="pattern">
        ([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?
      </param>
    </data>
  </define>
  <define name="ietf-inet-types__ipv6-address">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="ietf-inet-types__ip-prefix">
    <choice>
      <ref name="ietf-inet-types__ipv4-prefix"/>
      <ref name="ietf-inet-types__ipv6-prefix"/>
    </choice>
  </define>
  <define name="ietf-inet-types__host">
    <choice>
      <ref name="ietf-inet-types__ip-address"/>
      <ref name="ietf-inet-types__domain-name"/>
    </choice>
  </define>
  <define name="ietf-yang-types__date-and-time">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="__dhcp__subnet-list">
    <zeroOrMore>
      <element name="subnet">
        <element name="net">
          <ref name="ietf-inet-types__ip-prefix"/>
        </element>
        <interleave>
          <optional>
            <element name="range">
```

```
<interleave>
  <optional>
    <element name="dynamic-bootp">
      <empty/>
    </element>
  </optional>
  <element name="low">
    <ref name="ietf-inet-types__ip-address"/>
  </element>
  <element name="high">
    <ref name="ietf-inet-types__ip-address"/>
  </element>
</interleave>
</element>
</optional>
<optional>
  <element name="dhcp-options">
    <interleave>
      <zeroOrMore>
        <element name="router">
          <ref name="ietf-inet-types__host"/>
        </element>
      </zeroOrMore>
      <optional>
        <element name="domain-name">
          <ref name="ietf-inet-types__domain-name"/>
        </element>
      </optional>
    </interleave>
  </element>
</optional>
<optional>
  <element name="max-lease-time">
    <data type="unsignedInt"/>
  </element>
</optional>
</interleave>
</element>
</zeroOrMore>
</define>
<define name="ietf-inet-types__domain-name">
  <data type="string">
    <param name="pattern">... regex pattern ...</param>
    <param name="minLength">1</param>
    <param name="maxLength">253</param>
  </data>
</define>
<define name="ietf-inet-types__ipv4-prefix">
```

```

    <data type="string">
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="ietf-inet-types__ipv4-address">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="ietf-inet-types__ipv6-prefix">
    <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="pattern">... regex pattern ...</param>
    </data>
  </define>
  <define name="ietf-inet-types__ip-address">
    <choice>
      <ref name="ietf-inet-types__ipv4-address"/>
      <ref name="ietf-inet-types__ipv6-address"/>
    </choice>
  </define>
</grammar>

```

C.3.3. Schematron Schema for <nc: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:pattern abstract="true" id="_dhcp__subnet-list">
    <sch:rule context="$start/$pref:subnet">
      <sch:report test="preceding-sibling::$pref:subnet
        [$pref:net=current()/ $pref:net]">
        Duplicate key "net"
      </sch:report>
    </sch:rule>
    <sch:rule
      context="$start/$pref:subnet/$pref:dhcp-options/$pref:router">
      <sch:report test=".=preceding-sibling::router">
        Duplicate leaf-list value "<sch:value-of select="."/>"
      </sch:report>
    </sch:rule>
  </sch:pattern>
  <sch:pattern id="dhcp">
    <sch:rule
      context="/nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:default-lease-time">
      <sch:assert test="<sch:value-of select="."/> <sch:value-of select="."/> <sch:value-of select="."/>"
        The default-lease-time must be less than max-lease-time
    </sch:rule>
  </sch:pattern>
</sch:schema>

```



```
    </sch:assert>
  </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 "dhcp:name"
    </sch:report>
  </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 "dhcp:address"
    </sch:report>
  </sch:rule>
</sch:pattern>
<sch:pattern id="id2768196" is-a="_dhcp__subnet-list">
  <sch:param name="start" value="/nc:rpc-reply/nc:data/dhcp:dhcp"/>
  <sch:param name="pref" value="dhcp"/>
</sch:pattern>
<sch:pattern id="id2768215" is-a="_dhcp__subnet-list">
  <sch:param name="start"
    value="/nc:rpc-reply/nc:data/dhcp:dhcp/
      dhcp:shared-networks/dhcp:shared-network"/>
  <sch:param name="pref" value="dhcp"/>
</sch:pattern>
</sch:schema>
```

C.3.4. DSRL Schema for <nc:get> Reply

```
<?xml version="1.0" encoding="utf-8"?>
<dsrl:maps
  xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
  xmlns:dhcp="http://example.com/ns/dhcp"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data</dsrl:parent>
    <dsrl:name>dhcp:dhcp</dsrl:name>
    <dsrl:default-content>
      <dhcp:max-lease-time>7200</dhcp:max-lease-time>
      <dhcp:default-lease-time>600</dhcp:default-lease-time>
    </dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data/dhcp:dhcp</dsrl:parent>
    <dsrl:name>dhcp:max-lease-time</dsrl:name>
    <dsrl:default-content>7200</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data/dhcp:dhcp</dsrl:parent>
    <dsrl:name>dhcp:default-lease-time</dsrl:name>
    <dsrl:default-content>600</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:subnet
    </dsrl:parent>
    <dsrl:name>dhcp:max-lease-time</dsrl:name>
    <dsrl:default-content>7200</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:shared-networks/
      dhcp:shared-network/dhcp:subnet
    </dsrl:parent>
    <dsrl:name>dhcp:max-lease-time</dsrl:name>
    <dsrl:default-content>7200</dsrl:default-content>
  </dsrl:element-map>
</dsrl:maps>
```

Appendix D. Change Log

RFC Editor: remove this section upon publication as an RFC.

D.1. Changes Between Versions -07 and -08

- o Edits based on Gen-ART review.
- o Added formal templates in Section 13.
- o Created the "Contributors" section and moved the former co-authors there.
- o Indicated the location of both global and local named pattern definitions in the example hybrid schema in Section 8.1.
- o Added reference to EXSLT "evaluate" function.

D.2. Changes Between Versions -06 and -07

- o Mapping of 'description', 'reference' and 'units' to the hybrid schema is now mandatory.
- o Improvements and fixes of the text based on the AD review

D.3. Changes Between Versions -05 and -06

- o Terminology change: "conceptual tree schema" -> "hybrid schema".
- o Changed sectioning markers in the hybrid schema into plain NETMOD-specific annotations. Hence the former "nmt" namespace is not used at all.
- o Added the following NETMOD-specific annotations: @nma:if-feature, @nma:leaf-list, @nma:mandatory, @nma:module, removed @nma:presence.
- o Changed the structure of RELAX NG schemas by using embedded grammars and declaration of namespaces via @ns. This was necessary for enabling the "chameleon" behavior of global definitions.
- o Schematron validation phases are not used.
- o If an XPath expression appears inside a top-level grouping, the local prefix must be represented using the variable \$pref. (This is related to the previous item.)

- o DHCP example: All RNG schemas are only in the XML syntax. Added RNG with global definitions.
- o Added [XML-INFOSET] to normative references.
- o Listed the terms that are defined in other documents.
- o The schema for NETMOD-specific annotation is now given only as RNG named pattern definitions, no more in NVDL.

D.4. Changes Between Versions -04 and -05

- o Leafs that take their default value from a typedef and are not annotated with @nma:default must have @nma:implicit="true".
- o Changed code markers CODE BEGINS/ENDS to the form agreed by the WG.
- o Derived types "date-and-time" and "uri" SHOULD be mapped to XSD "dateTime" and "anyURI" types, respectively.
- o Clarified the notion of implicit nodes under 'case' in Section 9.1.2.
- o Moved draft-ietf-netmod-yang-types-06 to normative references.
- o An extra <rng:group> is no more required for the default case of a choice in the shorthand notation.

D.5. Changes Between Versions -03 and -04

- o Implemented ordering rules for list children - keys must go first and appear in the same order as in the input YANG module.
- o The 'case' statement is now mapped to either <rng:group> (inside RPC operations) or <rng:interleave> (otherwise).
- o A nma:default annotation coming from a datatype which the mapping expands is attached to the <rng:element> pattern where the expansion occurs. Added an example.
- o Documentation statements ('description', 'reference', 'status') MAY be ignored.
- o Single-valued numeric or length range parts are mapped to <rng:value> pattern or "length" facet.

- o Example for "string" datatype was added.
- o Appendix A now uses NVDL for defining NETMOD-specific annotations.
- o Added CODE BEGINS/ENDS markers.
- o Separated normative and informative references.
- o Added URL for XPath extensions namespace.
- o Added Section 2 (Terminology and Notation).
- o Added Section 14 (Security Considerations).
- o Added Section 16 (Acknowledgments).
- o Removed compact syntax schema from Appendix B.
- o Editorial changes: symbolic citation labels.

D.6. Changes Between Versions -02 and -03

- o Changed @nma:default-case to @nma:implicit.
- o Changed nma:leafref annotation from element to attribute.
- o Added skeleton rule to Section 11.2.
- o Reworked Section 11.3, added skeleton element maps, corrected the example.
- o Added section on 'feature' and 'deviation'.
- o New Section 9.1 integrating discussion of both optional/mandatory (was in -02) and implicit nodes (new).
- o Reflected that key argument and schema node identifiers are no more XPath (should be in yang-07).
- o Element patterns for implicit containers now must have @nma:implicit attribute.
- o Removed "float32" and "float64" types and added mapping of "decimal64" with example.
- o Removed mapping of 'require-instance' for "leafref" type.

- o Updated RELAX NG schema for NETMOD-specific annotations.
- o Updated the DHCP example.

D.7. Changes Between Versions -01 and -02

- o Moved Section 7 "NETCONF Content Validation" after Section 6.
- o New text about mapping defaults to DSRL, especially in Section 7 and Section 11.3.
- o Finished the DHCP example by adding the DSRL schema to Appendix C.
- o New @nma:presence annotation was added - it is needed for proper handling of default contents.
- o Section 11.2.1 "Constraints on Mandatory Choice" was added because these constraints require a combination of RELAX NG and Schematron.
- o Fixed the schema for NETMOD-specific annotations by adding explicit prefix to all defined elements and attributes. Previously, the attributes had no namespace.
- o Handling of 'feature', 'if-feature' and 'deviation' added.
- o Handling of nma:instance-identifier via XSLT extension function.

D.8. Changes Between Versions -00 and -01

- o Attributes @nma:min-elements and @nma:max-elements are attached to <rng:element> (list entry) and not to <rng:zeroOrMore> or <rng:oneOrMore>.
- o Keys and all node identifiers in 'key' and 'unique' statements are prefixed.
- o Fixed the mapping of 'rpc' and 'notification'.
- o Removed previous sec. 7.5 "RPC Signatures and Notifications" - the same information is now contained in Section 10.50 and Section 10.37.
- o Added initial "_" to mangled names of groupings.
- o Mandated the use of @xmlns:xxx as the only method for declaring the target namespace.

- o Added section "Handling of XML Namespaces" to explain the previous item.
- o Completed DHCP example in Appendix C.
- o Almost all text about the second mapping step is new.

Author's Address

Ladislav Lhotka (editor)
CESNET

Email: lhotka@cesnet.cz

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 5, 2011

A. Bierman
Brocade
October 2, 2010

Guidelines for Authors and Reviewers of YANG Data Model Documents
draft-ietf-netmod-yang-usage-11

Abstract

This memo provides guidelines for authors and reviewers of standards track specifications containing YANG data model modules. Applicable portions may be used as a basis for reviews of other YANG data model documents. Recommendations and procedures are defined, which are intended to increase interoperability and usability of Network Configuration Protocol (NETCONF) implementations which utilize YANG data model modules.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 5, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 4 |
| 2. Terminology | 5 |
| 2.1. Requirements Notation | 5 |
| 2.2. NETCONF Terms | 5 |
| 2.3. YANG Terms | 5 |
| 2.4. Terms | 6 |
| 3. General Documentation Guidelines | 7 |
| 3.1. Module Copyright | 7 |
| 3.2. Narrative Sections | 8 |
| 3.3. Definitions Section | 8 |
| 3.4. Security Considerations Section | 8 |
| 3.5. IANA Considerations Section | 9 |
| 3.5.1. Documents that Create a New Name Space | 9 |
| 3.5.2. Documents that Extend an Existing Name Space | 9 |
| 3.6. Reference Sections | 10 |
| 4. YANG Usage Guidelines | 11 |
| 4.1. Module Naming Conventions | 11 |
| 4.2. Identifiers | 11 |
| 4.3. Defaults | 11 |
| 4.4. Conditional Statements | 12 |
| 4.5. XPath Usage | 12 |
| 4.6. Lifecycle Management | 13 |
| 4.7. Module Header, Meta, and Revision Statements | 14 |
| 4.8. Namespace Assignments | 15 |
| 4.9. Top Level Data Definitions | 16 |
| 4.10. Data Types | 16 |
| 4.11. Reusable Type Definitions | 17 |
| 4.12. Data Definitions | 18 |
| 4.13. Operation Definitions | 19 |
| 4.14. Notification Definitions | 19 |
| 5. IANA Considerations | 20 |
| 6. Security Considerations | 21 |
| 6.1. Security Considerations Section Template | 21 |
| 7. Acknowledgments | 24 |
| 8. References | 25 |
| 8.1. Normative References | 25 |
| 8.2. Informative References | 25 |
| Appendix A. Module Review Checklist | 27 |
| Appendix B. YANG Module Template | 29 |
| Appendix C. Change Log | 32 |
| C.1. Changes from 10 to 11 | 32 |
| C.2. Changes from 09 to 10 | 32 |

| | | |
|------------------|-----------------------|----|
| C.3. | Changes from 08 to 09 | 32 |
| C.4. | Changes from 07 to 08 | 32 |
| C.5. | Changes from 06 to 07 | 32 |
| C.6. | Changes from 05 to 06 | 32 |
| C.7. | Changes from 04 to 05 | 33 |
| C.8. | Changes from 03 to 04 | 33 |
| C.9. | Changes from 02 to 03 | 34 |
| C.10. | Changes from 01 to 02 | 34 |
| C.11. | Changes from 00 to 01 | 34 |
| Author's Address | | 36 |

1. Introduction

The standardization of network configuration interfaces for use with the Network Configuration Protocol (NETCONF) [RFC4741] requires a modular set of data models, which can be reused and extended over time.

This document defines a set of usage guidelines for standards track documents containing YANG [I-D.ietf-netmod-yang] data models. YANG is used to define the data structures, protocol operations, and notification content used within a NETCONF server. A server which supports a particular YANG module will support client NETCONF operation requests, as indicated by the specific content defined in the YANG module.

This document is similar to the SMIV2 usage guidelines specification [RFC4181] in intent and structure. However, since that document was written a decade after SMIV2 modules had been in use, it was published as a 'best current practice' (BCP). This document is not a BCP, but rather an informational reference, intended to promote consistency in documents containing YANG modules.

Many YANG constructs are defined as optional to use, such as the description statement. However, in order to maximize interoperability of NETCONF implementations utilizing YANG data models, it is desirable to define a set of usage guidelines which may require a higher level of compliance than the minimum level defined in the YANG specification.

In addition, YANG allows constructs such as infinite length identifiers and string values, or top-level mandatory nodes, that a compliant server is not required to support. Only constructs which all servers are required to support can be used in IETF YANG modules.

This document defines usage guidelines related to the NETCONF operations layer, and NETCONF content layer, as defined in [RFC4741]. These guidelines are intended to be used by authors and reviewers to improve the readability and interoperability of published YANG data models.

2. Terminology

2.1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

RFC 2119 language is used here to express the views of the NETMOD working group regarding content for YANG modules. YANG modules complying with this document will treat the RFC 2119 terminology as if it were describing best current practices.

2.2. NETCONF Terms

The following terms are defined in [RFC4741] and are not redefined here:

- o capabilities
- o client
- o operation
- o server

2.3. YANG Terms

The following terms are defined in [I-D.ietf-netmod-yang] and are not redefined here:

- o data node
- o module
- o namespace
- o submodule
- o version
- o YANG
- o YIN

Note that the term 'module' may be used as a generic term for a YANG module or submodule. When describing properties which are specific to submodules, the term 'submodule' is used instead.

2.4. Terms

The following terms are used throughout this document:

published: A stable release of a module or submodule, usually contained in an RFC.

unpublished: An unstable release of a module or submodule, usually contained in an Internet-Draft.

3. General Documentation Guidelines

YANG data model modules under review are likely to be contained in Internet-Drafts. All guidelines for Internet-Draft authors MUST be followed. These guidelines are defined in [RFC2223] and updated in [RFC5741]. Additional information is also available online at:

<http://www.rfc-editor.org/rfc-editor/instructions2authors.txt>

The following sections MUST be present in an Internet-Draft containing a module:

- o Narrative sections
- o Definitions section
- o Security Considerations section
- o IANA Considerations section
- o References section

3.1. Module Copyright

The module description statement MUST contain a reference to the latest approved IETF Trust Copyright statement, which is available on-line at:

<http://trustee.ietf.org/license-info/>

Each YANG module or submodule contained within an Internet-Draft or RFC is considered to be a code component. The strings '<CODE BEGINS>' and '<CODE ENDS>' MUST be used to identify each code component.

The '<CODE BEGINS>' tag SHOULD be followed by a string identifying the file name specified in section 5.2 of [I-D.ietf-netmod-yang]. The following example is for the '2010-01-18' revision of the 'ietf-foo' module:


```
<CODE BEGINS> file "ietf-foo@2010-01-18.yang"
module ietf-foo {
    // ...
    revision 2010-01-18 {
        description "Latest revision";
        reference "RFC XXXXX";
    }
    // ...
}
<CODE ENDS>
```

Figure 1

3.2. Narrative Sections

The narrative part **MUST** include an overview section that describes the scope and field of application of the module(s) defined by the specification and that specifies the relationship (if any) of these modules to other standards, particularly to standards containing other YANG modules. The narrative part **SHOULD** include one or more sections to briefly describe the structure of the modules defined in the specification.

If the module(s) defined by the specification import definitions from other modules (except for those defined in the YANG [I-D.ietf-netmod-yang] or YANG Types [I-D.ietf-netmod-yang-types] documents), or are always implemented in conjunction with other modules, then those facts **MUST** be noted in the overview section, as **MUST** be noted any special interpretations of definitions in other modules.

3.3. Definitions Section

This section contains the module(s) defined by the specification. These modules **MUST** be written using the YANG syntax defined in [I-D.ietf-netmod-yang]. A YIN syntax version of the module **MAY** also be present in the document. There **MAY** also be other types of modules present in the document, such as SMIV2, which are not affected by these guidelines.

See Section 4 for guidelines on YANG usage.

3.4. Security Considerations Section

Each specification that defines one or more modules **MUST** contain a section that discusses security considerations relevant to those

modules. This section MUST be patterned after the latest approved template (available at <http://www.ops.ietf.org/netconf/yang-security-considerations.txt>).

In particular:

- o Writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be explained.
- o Readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.
- o Operations (i.e., YANG 'rpc' statements) which are potentially harmful to system behavior or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

3.5. IANA Considerations Section

In order to comply with IESG policy as set forth in <http://www.ietf.org/ID-Checklist.html>, every Internet-Draft that is submitted to the IESG for publication which has action items for IANA MUST contain an IANA Considerations section. The requirements for this section vary depending what actions are required of the IANA. If there are no IANA considerations applicable to the document, then the IANA Considerations section is not required. Refer to the guidelines in [RFC5226] for more details.

3.5.1. Documents that Create a New Name Space

If an Internet-Draft defines a new name space that is to be administered by the IANA, then the document MUST include an IANA Considerations section, that specifies how the name space is to be administered.

Specifically, if any YANG module namespace statement value contained in the document is not already registered with IANA, then a new YANG Namespace registry entry MUST be requested from the IANA. The YANG [I-D.ietf-netmod-yang] specification includes the procedure for this purpose in its IANA Considerations section.

3.5.2. Documents that Extend an Existing Name Space

It is possible to extend an existing namespace using a YANG submodule which belongs to an existing module already administered by IANA. In

this case, the document containing the main module MUST be updated to use the latest revision of the submodule.

3.6. Reference Sections

For every import or include statement which appears in a module contained in the specification, which identifies a module in a separate document, a corresponding normative reference to that document MUST appear in the Normative References section. The reference MUST correspond to the specific module version actually used within the specification.

For every normative reference statement which appears in a module contained in the specification, which identifies a separate document, a corresponding normative reference to that document SHOULD appear in the Normative References section. The reference SHOULD correspond to the specific document version actually used within the specification. If the reference statement identifies an informative reference, which identifies a separate document, a corresponding informative reference to that document MAY appear in the Informative References section.

4. YANG Usage Guidelines

In general, modules in IETF standards-track specifications MUST comply with all syntactic and semantic requirements of YANG. [I-D.ietf-netmod-yang]. The guidelines in this section are intended to supplement the YANG specification, which is intended to define a minimum set of conformance requirements.

In order to promote interoperability and establish a set of practices based on previous experience, the following sections establish usage guidelines for specific YANG constructs.

Only guidelines which clarify or restrict the minimum conformance requirements are included here.

4.1. Module Naming Conventions

Modules contained in standards track documents SHOULD be named according to the guidelines in the IANA considerations section of [I-D.ietf-netmod-yang].

A distinctive word or acronym (e.g., protocol name or working group acronym) SHOULD be used in the module name. If new definitions are being defined to extend one or more existing modules, then the same word or acronym should be reused, instead of creating a new one.

All published module names MUST be unique. For a YANG module published in an RFC, this uniqueness is guaranteed by IANA. For unpublished modules, the authors need to check that no other work in progress is using the same module name.

Once a module name is published, it MUST NOT be reused, even if the RFC containing the module is reclassified to 'Historic' status.

4.2. Identifiers

Identifiers for all YANG identifiers in published modules MUST be between 1 and 64 characters in length. These include any construct specified as an 'identifier-arg-str' token in the ABNF in section 12 of [I-D.ietf-netmod-yang].

4.3. Defaults

In general, it is suggested that sub-statements containing very common default values SHOULD NOT be present. The following sub-statements are commonly used with the default value, which would make the module difficult to read if used everywhere they are allowed.

| Statement | Default Value |
|--------------|---------------|
| config | true |
| mandatory | false |
| max-elements | unbounded |
| min-elements | 0 |
| ordered-by | system |
| status | current |
| yin-element | false |

4.4. Conditional Statements

A module may be conceptually partitioned in several ways, using the 'if-feature' and/or 'when' statements.

Data model designers need to carefully consider all modularity aspects, including the use of YANG conditional statements.

If a data definition is optional, depending on server support for a NETCONF protocol capability, then a YANG 'feature' statement SHOULD be defined to indicate that the NETCONF capability is supported within the data model.

If any notification data, or any data definition, for a non-configuration data node is not mandatory, then the server may or may not be required to return an instance of this data node. If any conditional requirements exist for returning the data node in a notification payload or retrieval request, they MUST be documented somewhere. For example, a 'when' or 'if-feature' statement could apply to the data node, or the conditional requirements could be explained in a 'description' statement within the data node or one of its ancestors (if any).

4.5. XPath Usage

This section describes guidelines for using the XML Path Language [W3C.REC-xpath-19991116] (XPath) within YANG modules.

The 'attribute' and 'namespace' axes are not supported in YANG, and MAY be empty in a NETCONF server implementation.

The 'position' and 'last' functions SHOULD NOT be used. This applies to implicit use of the 'position' function as well (e.g., '//chapter[42]'). A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'position' and 'last' functions MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

The 'preceding', and 'following' axes SHOULD NOT be used. These constructs rely on XML document order within a NETCONF server configuration database, which may not be supported consistently or produce reliable results across implementations. Predicate expressions based on static node properties (e.g., element name or value, 'ancestor' or 'descendant' axes) SHOULD be used instead. The 'preceding' and 'following' axes MAY be used if document order is not relevant to the outcome of the expression (e.g., check for global uniqueness of a parameter value.)

The 'preceding-sibling' and 'following-sibling' axes SHOULD NOT be used. A server is only required to maintain the relative XML document order of all instances of a particular user-ordered list or leaf-list. The 'preceding-sibling' and 'following-sibling' axes MAY be used if they are evaluated in a context where the context node is a user-ordered 'list' or 'leaf-list'.

Data nodes which use the 'int64' and 'uint64' built-in type SHOULD NOT be used within numeric expressions. There are boundary conditions in which the translation from the YANG 64-bit type to an XPath number can cause incorrect results. Specifically, an XPath 'double' precision floating point number cannot represent very large positive or negative 64-bit numbers because it only provides a total precision of 53 bits. The 'int64' and 'uint64' data types MAY be used in numeric expressions if the value can be represented with no more than 53 bits of precision.

Data modelers need to be careful not to confuse the YANG value space and the XPath value space. The data types are not the same in both, and conversion between YANG and XPath data types SHOULD be considered carefully.

Explicit XPath data type conversions MAY be used (e.g., 'string', 'boolean', or 'number' functions), instead of implicit XPath data type conversions.

4.6. Lifecycle Management

The status statement MUST be present if its value is 'deprecated' or 'obsolete'.

The module or submodule name MUST NOT be changed, once the document containing the module or submodule is published.

The module namespace URI value MUST NOT be changed, once the document containing the module is published.

The revision-date sub-statement within the imports statement SHOULD be present if any groupings are used from the external module.

The revision-date sub-statement within the include statement SHOULD be present if any groupings are used from the external sub-module.

If submodules are used, then the document containing the main module MUST be updated so that the main module revision date is equal or more recent than the revision date of any submodule which is (directly or indirectly) included by the main module.

4.7. Module Header, Meta, and Revision Statements

For published modules, the namespace MUST be a globally unique URI, as defined in [RFC3986]. This value is usually assigned by the IANA.

The organization statement MUST be present. If the module is contained in a document intended for standards-track status, then the organization SHOULD be the IETF working group chartered to write the document.

The contact statement MUST be present. If the module is contained in a document intended for standards-track status, then the working group WEB and mailing information MUST be present, and the main document author or editor contact information SHOULD be present. If additional authors or editors exist, their contact information MAY be present. In addition, the Area Director and other contact information MAY be present.

The description statement MUST be present. The appropriate IETF Trust Copyright text MUST be present, as described in Section 3.1.

If the module relies on information contained in other documents, which are not the same documents implied by the import statements present in the module, then these documents MUST be identified in the reference statement.

A revision statement MUST be present for each published version of the module. The revision statement MUST have a reference substatement. It MUST identify the published document which contains the module. Modules are often extracted from their original documents and it is useful for developers and operators to know how

to find the original source document in a consistent manner. The revision statement MAY have a description substatement.

Each new revision MUST include a revision date which is higher than any other revision date in the module. The revision date does not need to be updated if the module contents do not change in the new document revision.

It is acceptable to reuse the same revision statement within unpublished versions (i.e., Internet-Drafts), but the revision date MUST be updated to a higher value each time the Internet-Draft is re-published.

4.8. Namespace Assignments

It is RECOMMENDED that only valid YANG modules are included in documents, whether they are published yet or not. This allows:

- o the module to compile correctly instead of generating disruptive fatal errors.
- o early implementors to use the modules without picking a random value for the XML namespace.
- o early interoperability testing since independent implementations will use the same XML namespace value.

Until a URI is assigned by the IANA, a proposed namespace URI MUST be provided for the namespace statement in a YANG module. A value SHOULD be selected which is not likely to collide with other YANG namespaces. Standard module names, prefixes, and URI strings already listed in the YANG Module Registry MUST NOT be used.

A standard namespace statement value SHOULD have the following form:

```
<URN prefix string>:<module-name>
```

The following URN prefix string SHOULD be used for published and unpublished YANG modules:

```
urn:ietf:params:xml:ns:yang:
```

The following example URNs would be valid temporary namespace statement values for standards-track modules:

```
urn:ietf:params:xml:ns:yang:ietf-netconf-partial-lock
```



```
urn:ietf:params:xml:ns:yang:ietf-netconf-state
```

```
urn:ietf:params:xml:ns:yang:ietf-netconf
```

Note that a different URN prefix string SHOULD be used for non-standards track modules. The string SHOULD be selected according to the guidelines in [I-D.ietf-netmod-yang].

The following examples of non-standards track modules are only suggestions. There are no guidelines for this type of URN in this document:

```
http://example.com/ns/example-interfaces
```

```
http://example.com/ns/example-system
```

4.9. Top Level Data Definitions

There SHOULD only be one top-level data node defined in each YANG module, if any data nodes are defined at all.

The top-level data organization SHOULD be considered carefully, in advance. Data model designers need to consider how the functionality for a given protocol or protocol family will grow over time.

The names and data organization SHOULD reflect persistent information, such as the name of a protocol. The name of the working group SHOULD NOT be used because this may change over time.

A mandatory database data definition is defined as a node that a client must provide for the database to be valid. The server is not required to provide a value.

Top-level database data definitions MUST NOT be mandatory. If a mandatory node appears at the top-level, it will immediately cause the database to be invalid. This can occur when the server boots or when a module is loaded dynamically at runtime.

4.10. Data Types

Selection of an appropriate data type (i.e., built-in type, existing derived type, or new derived type) is very subjective and therefore few requirements can be specified on that subject.

Data model designers SHOULD use the most appropriate built-in data type for the particular application.

If extensibility of enumerated values is required, then the

'identityref' data type SHOULD be used instead of an enumeration or other built-in type.

For string data types, if a machine-readable pattern can be defined for the desired semantics, then one or more pattern statements SHOULD be present.

For string data types, if the length of the string is required to be bounded in all implementations, then a length statement MUST be present.

For numeric data types, if the values allowed by the intended semantics are different than those allowed by the unbounded intrinsic data type (e.g., 'int32'), then a range statement SHOULD be present.

The signed numeric data types (i.e., 'int8', 'int16', 'int32', and 'int64') SHOULD NOT be used unless negative values are allowed for the desired semantics.

For 'enumeration' or 'bits' data types, the semantics for each 'enum' or 'bit' SHOULD be documented. A separate description statement (within each 'enum' or 'bit' statement) SHOULD be present.

4.11. Reusable Type Definitions

If an appropriate derived type exists in any standard module, such as [I-D.ietf-netmod-yang-types], then it SHOULD be used instead of defining a new derived type.

If an appropriate units identifier can be associated with the desired semantics, then a units statement SHOULD be present.

If an appropriate default value can be associated with the desired semantics, then a default statement SHOULD be present.

If a significant number of derived types are defined, and it is anticipated that these data types will be reused by multiple modules, then these derived types SHOULD be contained in a separate module or submodule, to allow easier reuse without unnecessary coupling.

The description statement MUST be present.

If the type definition semantics are defined in an external document (other than another YANG module indicated by an import statement), then the reference statement MUST be present.

4.12. Data Definitions

The description statement MUST be present in the following YANG statements:

- o anyxml
- o augment
- o choice
- o container
- o extension
- o feature
- o grouping
- o identity
- o leaf
- o leaf-list
- o list
- o notification
- o rpc
- o typedef

If the data definition semantics are defined in an external document, (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

The 'anyxml' construct may be useful to represent an HTML banner containing markup elements, such as '' and '', and MAY be used in such cases . However, this construct SHOULD NOT be used if other YANG data node types can be used instead to represent the desired syntax and semantics.

If there are referential integrity constraints associated with the desired semantics that can be represented with XPath, then one or more must statements SHOULD be present.

For list and leaf-list data definitions, if the number of possible

instances is required to be bounded for all implementations, then the max-elements statements SHOULD be present.

If any must or when statements are used within the data definition, then the data definition description statement SHOULD describe the purpose of each one.

4.13. Operation Definitions

If the operation semantics are defined in an external document (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

If the operation impacts system behavior in some way, it SHOULD be mentioned in the description statement.

If the operation is potentially harmful to system behavior in some way, it MUST be mentioned in the Security Considerations section of the document.

4.14. Notification Definitions

The description statement MUST be present.

If the notification semantics are defined in an external document (other than another YANG module indicated by an import statement), then a reference statement MUST be present.

5. IANA Considerations

This document registers one URI in the IETF XML registry [RFC3688]. The following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-template

Registrant Contact: The NETMOD WG of the IETF.

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

This document requests the following assignment in the YANG Module Names Registry for the YANG module template in Appendix B.

| Field | Value |
|-----------|---|
| name | ietf-template |
| namespace | urn:ietf:params:xml:ns:yang:ietf-template |
| prefix | temp |
| reference | RFCXXXX |

6. Security Considerations

This document defines documentation guidelines for NETCONF content defined with the YANG data modeling language. The guidelines for how to write a Security Considerations section for a YANG module are defined in the online document

<http://www.ops.ietf.org/netconf/yang-security-considerations.txt>

This document does not introduce any new or increased security risks into the management system.

The following section contains the security considerations template dated 2010-06-16. Be sure to check the WEB page at the URL listed above in case there is a more recent version available.

Each specification that defines one or more YANG modules MUST contain a section that discusses security considerations relevant to those modules. This section MUST be patterned after the latest approved template (available at [ed: URL TBD]).

In particular, writable data nodes that could be especially disruptive if abused MUST be explicitly listed by name and the associated security risks MUST be spelled out.

Similarly, readable data nodes that contain especially sensitive information or that raise significant privacy concerns MUST be explicitly listed by name and the reasons for the sensitivity/privacy concerns MUST be explained.

Further, if new RPC operations have been defined, then the security considerations of each new RPC operation MUST be explained.

6.1. Security Considerations Section Template

X. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC4741]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH [RFC4742].

```
-- if you have any writeable data nodes (those are all the
-- "config true" nodes, and remember, that is the default)
-- describe their specific sensitivity or vulnerability.
```

There are a number of data nodes defined in this YANG module which are writable/creatable/deletable (i.e. config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g. edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

<list subtrees and data nodes and state why they are sensitive>

```
-- for all YANG modules you must evaluate whether any readable data
-- nodes (those are all the "config false" nodes, but also all other
-- nodes, because they can also be read via operations like get or
-- get-config) are sensitive or vulnerable (for instance, if they
-- might reveal customer information or violate personal privacy
-- laws such as those of the European Union if exposed to
-- unauthorized parties)
```

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g. via get, get-config or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

<list subtrees and data nodes and state why they are sensitive>

```
-- if your YANG module has defined any rpc operations
-- describe their specific sensitivity or vulnerability.
```

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

<list RPC operations and state why they are sensitive>

Figure 2

7. Acknowledgments

The structure and contents of this document are adapted from Guidelines for MIB Documents [RFC4181], by C. M. Heard.

The working group thanks Martin Bjorklund and Juergen Schoenwaelder for their extensive reviews and contributions to this document.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2223] Postel, J. and J. Reynolds, "Instructions to RFC Authors", RFC 2223, October 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.
- [RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.
- [RFC5741] Daigle, L., Kolkman, O., and IAB, "RFC Streams, Headers, and Boilerplates", RFC 5741, December 2009.
- [W3C.REC-xpath-19991116]
DeRose, S. and J. Clark, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.
- [I-D.ietf-netmod-yang]
Bjorklund, M., "YANG - A data modeling language for the Network Configuration Protocol (NETCONF)", draft-ietf-netmod-yang-13 (work in progress), June 2010.
- [I-D.ietf-netmod-yang-types]
Schoenwaelder, J., "Common YANG Data Types", draft-ietf-netmod-yang-types-09 (work in progress), April 2010.

8.2. Informative References

- [RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB Documents", BCP 111, RFC 4181, September 2005.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an

IANA Considerations Section in RFCs", BCP 26, RFC 5226,
May 2008.

Appendix A. Module Review Checklist

This section is adapted from RFC 4181.

The purpose of a YANG module review is to review the YANG module both for technical correctness and for adherence to IETF documentation requirements. The following checklist may be helpful when reviewing a draft document:

1. I-D Boilerplate -- verify that the draft contains the required Internet-Draft boilerplate (see <http://www.ietf.org/ietf/lid-guidelines.txt>), including the appropriate statement to permit publication as an RFC, and that I-D boilerplate does not contain references or section numbers.
2. Abstract -- verify that the abstract does not contain references, that it does not have a section number, and that its content follows the guidelines in <http://www.ietf.org/ietf/lid-guidelines.txt>.
3. IETF Trust Copyright -- verify that the draft has the appropriate text regarding the rights that document contributors provide to the IETF Trust [RFC5378]. Some guidelines related to this requirement are described in Section 3.1. The IETF Trust license policy (TLP) can be found at:

<http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf>
4. Security Considerations Section -- verify that the draft uses the latest approved template from the OPS area web site (<http://www.ops.ietf.org/netconf/yang-security-considerations.txt>) and that the guidelines therein have been followed.
5. IANA Considerations Section -- this section must always be present. For each module within the document, ensure that the IANA Considerations section contains entries for the following IANA registries:

XML Namespace Registry: Register the YANG module namespace.

YANG Module Registry: Register the YANG module name, prefix, namespace, and RFC number, according to the rules specified in [I-D.ietf-netmod-yang].
6. References -- verify that the references are properly divided between normative and informative references, that RFC 2119 is included as a normative reference if the terminology defined therein is used in the document, that all references required by

the boilerplate are present, that all YANG modules containing imported items are cited as normative references, and that all citations point to the most current RFCs unless there is a valid reason to do otherwise (for example, it is OK to include an informative reference to a previous version of a specification to help explain a feature included for backward compatibility). Be sure citations for all imported modules are present somewhere in the document text (outside the YANG module).

7. Copyright Notices -- verify that the draft contains an abbreviated IETF Trust copyright notice in the description statement of each YANG module or sub-module, and that it contains the full IETF Trust copyright notice at the end of the document. Make sure that the correct year is used in all copyright dates. Use the approved text from the latest Trust Legal Provisions (TLP) document, which can be found at:

<http://trustee.ietf.org/license-info/>

8. Other Issues -- check for any issues mentioned in <http://www.ietf.org/ID-Checklist.html> that are not covered elsewhere.

9. Technical Content -- review the actual technical content for compliance with the guidelines in this document. The use of a YANG module compiler is recommended when checking for syntax errors. A list of freely available tools and other information can be found at:

<http://trac.tools.ietf.org/wg/netconf/trac/wiki>

Checking for correct syntax, however, is only part of the job. It is just as important to actually read the YANG module document from the point of view of a potential implementor. It is particularly important to check that description statements are sufficiently clear and unambiguous to allow interoperable implementations to be created.

Appendix B. YANG Module Template

```
<CODE BEGINS> file "ietf-template@2010-05-18.yang"

module ietf-template {

    // replace this string with a unique namespace URN value
    namespace
        "urn:ietf:params:xml:ns:yang:ietf-template";

    // replace this string, and try to pick a unique prefix
    prefix "temp";

    // import statements here: e.g.,
    // import ietf-yang-types { prefix yang; }
    // import ietf-inet-types { prefix inet; }

    // identify the IETF working group if applicable
    organization
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    // update this contact statement with your info
    contact
        "WG Web:    <http://tools.ietf.org/wg/your-wg-name/>
        WG List:    <mailto:your-wg-name@ietf.org>

        WG Chair:  your-WG-chair
                   <mailto:your-WG-chair@example.com>

        Editor:    your-name
                   <mailto:your-email@example.com>";

    // replace the first sentence in this description statement.
    // replace the copyright notice with the most recent
    // version, if it has been updated since the publication
    // of this document
    description
        "This module defines a template for other YANG modules.

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

        Redistribution and use in source and binary forms, with or
```

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

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

// RFC Ed.: replace XXXX with actual RFC number and remove this note

reference "RFC XXXX";

// RFC Ed.: remove this note

// Note: extracted from draft-ietf-netmod-yang-usage-04.txt

// replace '2010-05-18' with the module publication date

// The format is (year-month-day)

```
revision "2010-05-18" {  
  description  
    "Initial version";  
}
```

// extension statements

// feature statements

// identity statements

// typedef statements

// grouping statements

// data definition statements

// augment statements

// rpc statements

// notification statements

// DO NOT put deviation statements in a published module

}

<CODE ENDS>

Figure 3

Appendix C. Change Log

C.1. Changes from 10 to 11

- o Removed Intellectual Property section, since no longer required.
- o Reworded XPath guidelines related to XML document order, 'int64' and 'uint64' data types, and 'anyxml' data nodes.

C.2. Changes from 09 to 10

- o Added security considerations section template.
- o Added guideline for documenting conditional requirements for non-mandatory non-configuration data nodes.
- o Clarified that revision date update applies to the module contents.

C.3. Changes from 08 to 09

- o Clarifications and corrections to address Gen-ART review comments.

C.4. Changes from 07 to 08

- o Corrected YANG security considerations URL.
- o Expanded 'CODE BEGINS' example.
- o Added RPC operations to the security considerations guidelines section.
- o Removed guideline about leading and trailing whitespace.

C.5. Changes from 06 to 07

- o Corrected title change bug; supposed to be page header instead.
- o Fixed typos added to last revision.
- o Added sentence to checklist to make sure text outside module contains citations for imports.

C.6. Changes from 05 to 06

- o Several clarifications and corrections, based on the AD review by Dan Romascanu.

C.7. Changes from 04 to 05

- o Changed 'object' terminology to 'data definition'.
- o Put XPath guidelines in separate section.
- o Clarified XPath usage for XML document order dependencies.
- o Updated <CODE BEGINS> guidelines to current conventions.
- o Added informative reference for IANA considerations guidelines and XML registry.
- o Updated IANA Considerations section to reserve the ietf-template module in the YANG Module Name Registry so it cannot be reused accidentally.
- o Many other clarifications and fixed typos found in WGLC reviews.

C.8. Changes from 03 to 04

- o Removed figure 1 to reduce duplication, just refer to 4741bis draft.
- o Fixed bugs and typos found in WGLC reviews.
- o Removed some guidelines and referring to YANG draft instead of duplicating YANG rules here.
- o Changed security guidelines so they refer to the IETF Trust TLP instead of MIB-specific references.
- o Change temporary namespace guidelines so the DRAFT-XX and RFC-nnnn suffix strings are not used.
- o Changed some MIB boilerplate so it refers to YANG boilerplate instead.
- o Introduced dangling URL reference to online YANG security guidelines

<http://www.ops.ietf.org/yang-security.html>

[ed.: Text from Bert Wijnen will be completed soon and posted online, and then this URL will be finalized.]

- o Moved reference for identifying the source document inside the each revision statement.

- o Removed guideline about valid XPath since YANG already requires valid XPath.
- o Added guideline that strings should not rely on preservation of leading and trailing whitespace characters.
- o Relaxed some XPath and anyxml guidelines from SHOULD NOT or MUST NOT to MAY use with caution.
- o Updated the TLP text within the example module again.
- o Reversed order of change log so most recent entries are first.

C.9. Changes from 02 to 03

- o Updated figure 1 to align with 4741bis draft.
- o Updated guidelines for import-by-revision and include-by-revision.
- o Added file name to code begins convention in ietf-template module.

C.10. Changes from 01 to 02

- o Updated figure 1 per mailing list comments.
- o Updated suggested organization to include the working group name.
- o Updated ietf-template.yang to use new organization statement value.
- o Updated Code Component requirements as per new TLP.
- o Updated ietf-template.yang to use new Code Component begin and end markers.
- o Updated references to the TLP in a couple sections.
- o Change manager/agent terminology to client/server.

C.11. Changes from 00 to 01

- o Added transport 'TLS' to figure 1.
- o Added note about RFC 2119 terminology.
- o Corrected URL for instructions to authors.

- o Updated namespace procedures section.
- o Updated guidelines on module contact, reference, and organization statements.
- o Added note on use of preceding-sibling and following-sibling axes in XPath expressions.
- o Added section on temporary namespace statement values.
- o Added section on top level database objects.
- o Added ietf-template.yang appendix.

Author's Address

Andy Bierman
Brocade

Email: andy.bierman@brocade.com

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 22, 2011

B. Linowski
TCS/Nokia Siemens Networks
M. Ersue
Nokia Siemens Networks
S. Kuryla
360 Treasury Systems
October 19, 2010

Extending YANG with Language Abstractions
draft-linowski-netmod-yang-abstract-04

Abstract

YANG - the NETCONF Data Modeling Language - supports modeling of a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF. This memo suggests to enhance YANG with supplementary modeling features and language abstractions with the aim to improve the model extensibility and reuse.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 22, 2011.

Copyright Notice

Copyright (c) 2010 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 5 |
| 1.1. Key Words | 5 |
| 1.2. Motivation | 5 |
| 1.3. Modeling Improvements with Language Abstractions | 6 |
| 1.4. Design Approach | 8 |
| 1.5. Modeling Resource Models with YANG | 8 |
| 1.5.1. Example of a Physical Network Resource Model | 8 |
| 1.5.2. Modeling Entity MIB Entries as Physical Resources | 12 |
| 2. Complex Types | 16 |
| 2.1. Definition | 16 |
| 2.2. complex-type extension statement | 16 |
| 2.3. instance extension statement | 18 |
| 2.4. instance-list extension statement | 19 |
| 2.5. extends extension statement | 20 |
| 2.6. abstract extension statement | 20 |
| 2.7. XML Encoding Rules | 21 |
| 2.8. Type Encoding Rules | 21 |
| 2.9. Extension and Feature Definition Module | 22 |
| 2.10. Example Model for Complex Types | 25 |
| 2.11. NETCONF Payload Example | 26 |
| 2.12. Update Rules for Modules Using Complex Types | 29 |
| 2.13. Using Complex Types | 29 |
| 2.13.1. Overriding Complex Type Data Nodes | 29 |
| 2.13.2. Augmenting Complex Types | 30 |
| 2.13.3. Controlling the Use of Complex Types | 31 |
| 3. Typed Instance Identifier | 32 |
| 3.1. Definition | 32 |
| 3.2. instance-type extension statement | 32 |
| 3.3. Typed Instance Identifier Example | 32 |
| 4. IANA Considerations | 33 |
| 5. Security Considerations | 34 |
| 6. Acknowledgements | 34 |
| 7. References | 34 |
| 7.1. Normative References | 34 |
| 7.2. Informative References | 35 |
| Appendix A. Change Log | 35 |
| A.1. 03-04 | 35 |
| A.2. 02-03 | 36 |
| A.3. 01-02 | 36 |
| A.4. 00-01 | 37 |
| Appendix B. YANG Modules for Physical Network Resource Model and Hardware Entities Model | 37 |
| Appendix C. Example YANG Module for the IPFIX/PSAMP Model | 44 |
| C.1. Modeling Improvements for the IPFIX/PSAMP Model with Complex types and Typed instance identifiers | 44 |
| C.2. IPFIX/PSAMP Model with Complex Types and Typed | |

| | |
|--------------------------------|----|
| Instance Identifiers | 45 |
|--------------------------------|----|

1. Introduction

YANG - the NETCONF Data Modeling Language ([RFC6020]) - supports modeling of a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF. This document defines extensions for the modeling language YANG as new language statements, which introduce language abstractions to improve the model extensibility and reuse. A model example from an actual network management system is given to highlight the value of proposed language extensions, especially class inheritance and recursiveness. The language extensions defined in this document have been implemented with two open source tools. These tools have been used to validate the model examples through the document.

1.1. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.2. Motivation

- o Many systems today have a management information base that in effect is organized as a tree build of recursively nested container nodes. For example, the physical resources in the ENTITY-MIB conceptually form a containment tree. The index entPhysicalContainedIn points to the containing entity in a flat list. The ability to represent nested, recursive data structures of arbitrary depth would enable the representation of the primary containment hierarchy of physical entities as a node tree in the server MIB and in the NETCONF payload.
- o A manager scanning the network in order to update the state of an inventory management system might be only interested in data structures that represent a specific type of hardware. Such a manager would then look for entities that are of this specific type, including those that are an extension or specialization of this type. To support this use case, it is helpful to bear the corresponding type information within the data structures, which describe the network element hardware.
- o A system that is managing network elements is concerned e.g. with managed objects of type "plug-in modules" that have a name, a version and an activation state. In this context, it is useful to define the "plug-in module" as a concept that is supposed to be further detailed and extended by additional concrete model

elements. In order to realize such a system, it is worth to model abstract entities, which enable reuse and ease concrete refinements of that abstract entity in a second step.

- o As particular network elements have specific type of components that need to be managed (OS images, plug-in modules, equipment, etc.), it should be possible to define concrete types, which describe the managed object precisely. By using type-safe extensions of basic concepts a system in the manager role can safely and explicitly determine that e.g. the "equipment" is actually of type "network card".
- o Currently different SDOs are working on the harmonization of their management information models. Often a model mapping or transformation between systems becomes necessary. The harmonization of the models is done e.g. by mapping of the two models on object level or integrating an object hierarchy into an existing information model. Extending YANG with language abstractions can simplify on the one hand the adoption of IETF resource models by other SDOs and facilitate the alignment with other SDO's resource models (e.g. TM Forum SID). The proposed YANG extensions can on the other hand enable the utilization of YANG modeling language in other SDOs, which are used to model complex management systems in a top-down manner and use high-level language features frequently.

This memo specifies additional modeling features for the YANG language in the area of structured model abstractions, typed references as well as recursive data structures and discusses how these new features can improve the modeling capabilities of YANG.

Section 1.5.1 contains a physical resource model, which deals with some of the modeling challenges illustrated above. Section 1.5.2 gives an example, which uses the base classes defined in the physical resource model and derives a model for physical entities defined in Entity MIB".

1.3. Modeling Improvements with Language Abstractions

Complex Types and Typed Instance Identifiers provide various technical improvements on modeling level:

- o In case the model of a system that should be managed with NETCONF makes use of inheritance, complex types enable an almost one-to-one mapping between the classes in the original model and the YANG module.

- o Typed instance identifiers allow representing associations between the concepts in a type-safe way to prevent type errors caused by referring to data nodes of incompatible types. This avoids referring to a particular location in the MIB, which is not mandated by the domain model.
- o Complex types allow defining complete, self-contained type definitions. It is not necessary to explicitly add a key statement to lists, which use a grouping defining the data nodes.
- o Complex types simplify concept refinement by extending a base complex type and make it superfluous to represent concept refinements with workarounds such as huge choice-statements with complex branches.
- o Abstract complex types ensure correct usage of abstract concepts by enforcing the refinement of common set of properties before instantiation.
- o Complex types allow defining recursive structures. This enables to represent complex structures of arbitrary depth by nesting instances of basic complex types that may contain themselves.
- o Complex types avoid introducing meta-data types (e.g. type code enumerations) and meta-data leafs (e.g. leafs containing a type code) to indicate, which concrete type of object is actually represented by a generic container in the MIB. This also avoids to explicitly rule out illegal use of sub-type specific properties in generic containers.
- o Complex type instances include the type information in the NETCONF payload. This allows to determine the actual type of an instance during the NETCONF payload parsing and avoids the use of additional leafs in the model, which provide the type information as content.
- o Complex types may be declared explicitly as optional features, which is not possible when the actual type of an entity represented by a generic container is indicated with a type code enumeration.

Appendix C 'Example YANG Module for the IPFIX/PSAMP Model' lists technical improvements for modeling with Complex Types and Typed Instance Identifiers and exemplifies the usage of the proposed YANG extensions based on the IPFIX/PSAMP configuration model in [IPFIXCONF].

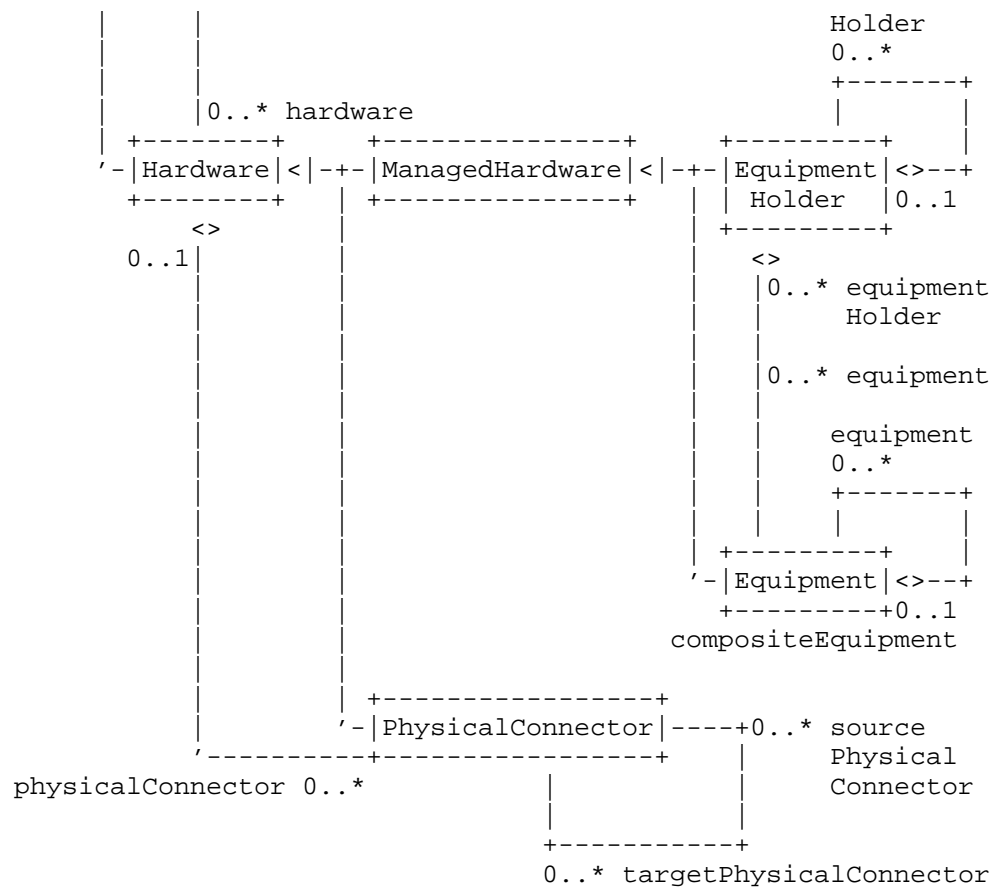


Figure 1: Physical Network Resource Model

Since this model is an abstraction of network element specific MIB topologies, modeling it with YANG creates some challenges. Some of these challenges and how they can be addressed with complex types are explained below:

- o Modeling of abstract concepts: Classes like "Resource" represent concepts that primarily serve as a base class for derived classes. With complex types, such an abstract concept could be represented by an abstract complex type (see "complex-type extension statement" and "abstract extension statement").
- o Class Inheritance: Information models for complex management domains often use class inheritance to create specialized classes

like "PhysicalConnector" from a more generic base class (here "Hardware"), which itself might inherit from another base class ("PhysicalResource") etc. Complex types allow creating enhanced versions of an existing (abstract or concrete) base type via an extension (see "extends extension statement").

- o Recursive containment: In order to specify containment hierarchies models frequently contain different aggregation associations, in which the target (contained element) is either the containing class itself or a base class of the containing class. In the model above, the recursive containment of "EquipmentHolder" is an example of such a relationship. Complex types support such a containment by using a complex type (or one of its ancestor types) as type of an instance or instance list that is part of its definition (see "instance(-list) extension statement").
- o Reference relationships: A key requirement on large models for network domains with many related managed objects is the association between classes that represent an essential relationship between instances of such a class. For example, the relationship between "PhysicalLink" and "Hardware" tells which physical link is connecting which hardware resources. It is important to notice that this kind of relationships do not mandate any particular location of the two connected hardware instances in any MIB. Such containment agnostic relationships can be represented by a typed instance identifier that embodies one direction of such an association (see "Typed instance identifiers").

The YANG module excerpt below shows how the challenges listed above can be addressed by the Complex Types extension (module import prefix "ct:"). The complete YANG module for the physical resource model in Figure 1 can be found in Appendix B: 'YANG Modules for Physical Network Resource Model and Hardware Entities Model'.

Note: The YANG extensions proposed in this document have been implemented as the open source tools "Pyang Extension for Complex Types" ([Pyang-ct], ([Pyang]) and "Libsmi Extension for Complex Types" ([Libsmi]). All model examples in the document have been validated with the tools Pyang-ct and Libsmi.

<CODE BEGINS>

```
module udmcore {  
  
    namespace "http://example.com/udmcore";
```



```
prefix "udm";

import ietf-complex-types {prefix "ct"; }

    // Basic complex types...

ct:complex-type PhysicalResource {
  ct:extends Resource;
  ct:abstract true;
  // ...
  leaf serialNumber {type string;}
}

ct:complex-type Hardware {
  ct:extends PhysicalResource;
  ct:abstract true;
  // ...
  leaf-list physicalLink {
    type instance-identifier {ct:instance-type PhysicalLink;}
  }
  ct:instance-list containedHardware {
    ct:instance-type Hardware;
  }
}

ct:instance-list physicalConnector {
  ct:instance-type PhysicalConnector;
}

ct:complex-type PhysicalLink {
  ct:extends PhysicalResource;
  // ...
  leaf-list hardware {
    type instance-identifier {ct:instance-type Hardware;}
  }
}

ct:complex-type ManagedHardware {
  ct:extends Hardware;
  ct:abstract true;
  // ...
}

ct:complex-type PhysicalConnector {
  ct:extends Hardware;
```

```
        leaf location {type string;}
        // ...
    leaf-list sourcePhysicalConnector {
        type instance-identifier {ct:instance-type PhysicalConnector;}
    }
    leaf-list targetPhysicalConnector {
        type instance-identifier {ct:instance-type PhysicalConnector;}
    }
}

ct:complex-type Equipment {
    ct:extends ManagedHardware;
    // ...
    ct:instance-list equipment {
        ct:instance-type Equipment;
    }
}

ct:complex-type EquipmentHolder {
    ct:extends ManagedHardware;
    leaf vendorName {type string;}
    // ...
    ct:instance-list equipment {
        ct:instance-type Equipment;
    }
    ct:instance-list equipmentHolder {
        ct:instance-type EquipmentHolder;
    }
}
// ...
}

<CODE ENDS>
```

1.5.2. Modeling Entity MIB Entries as Physical Resources

The physical resource module described above can now be used to model physical entities as defined in the Entity MIB [RFC4133]. For each physical entity class listed in the "PhysicalClass" enumeration, a complex type is defined. Each of these complex types extends the most specific complex type already available in the physical resource module. For example, the type "HWModule" extends the complex type "Equipment" as a hardware module. Physical entity properties that should be included in a physical entity complex type are combined in a grouping, which is then used in each complex type definition of an entity.

This approach has following benefits:

- o The definition of the complex types for hardware entities becomes compact as many of the features can be reused from the basic complex type definition.
- o Physical entities are modeled in a consistent manner as predefined concepts are extended.
- o Entity MIB specific attributes as well as vendor specific attributes can be added without having to define separate extension data nodes.

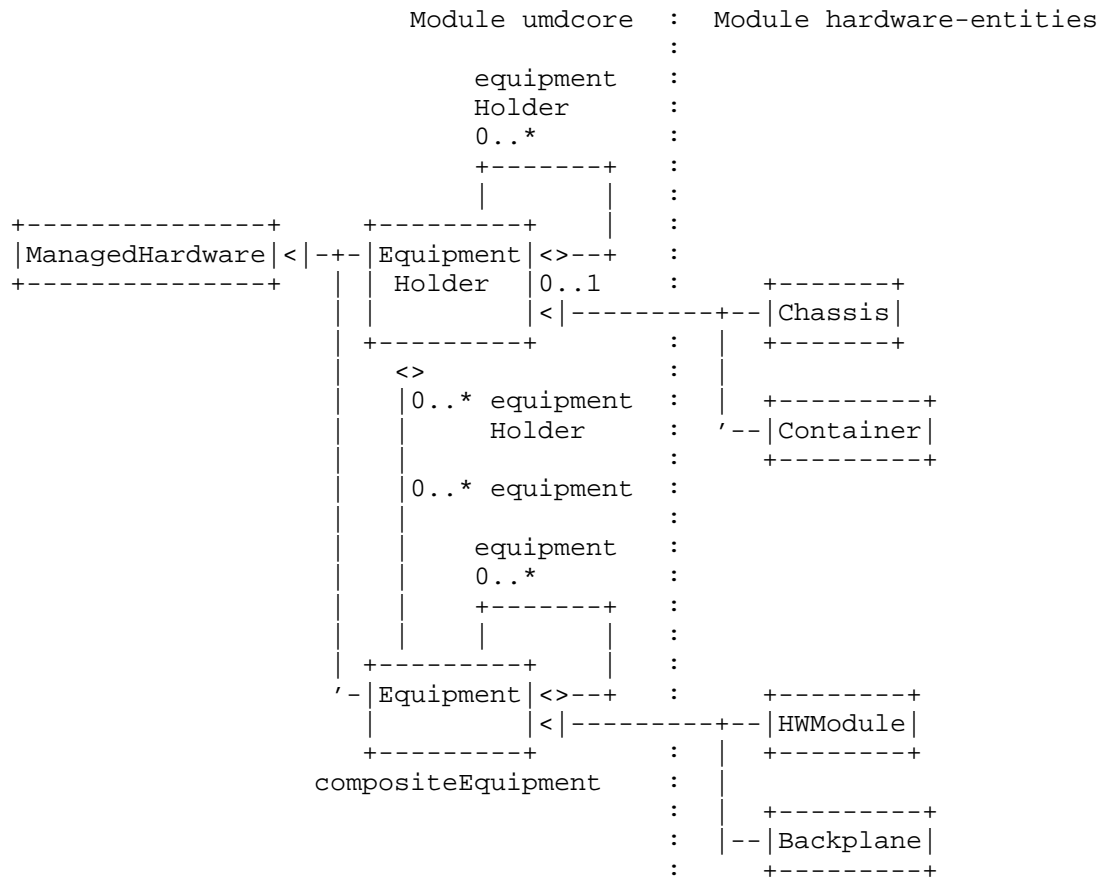


Figure 2: Hardware Entities Model

Below is an excerpt of the according YANG module using complex types to model hardware entities. The complete YANG module for the Hardware Entities model in Figure 2 can be found in Appendix B: 'YANG Modules for Physical Network Resource Model and Hardware Entities Model'.

<CODE BEGINS>

```
module hardware-entities {

    namespace "http://example.com/hardware-entities";
    prefix "hwe";

    import ietf-yang-types {prefix "yt";}
    import ietf-complex-types {prefix "ct";}
    import udmcore {prefix "uc";}

    grouping PhysicalEntityProperties {
        // ...
        leaf mfgDate {type yang:date-and-time; }
        leaf-list uris {type string; }
    }

    // Physical entities representing equipment

    ct:complex-type HWModule {
        ct:extends uc:Equipment;
        description "Complex type representing module entries
                    (entPhysicalClass = module(9)) in entPhysicalTable";
        uses PhysicalEntityProperties;
    }

    // ...

    // Physical entities representing equipment holders

    ct:complex-type Chassis {
        ct:extends uc:EquipmentHolder;
        description "Complex type representing chassis entries
                    (entPhysicalClass = chassis(3)) in entPhysicalTable";
        uses PhysicalEntityProperties;
    }

    // ...
}

<CODE ENDS>
```

2. Complex Types

2.1. Definition

YANG type concept is currently restricted to simple types, e.g. restrictions of primitive types, enumerations or union of simple types.

Complex types are types with a rich internal structure, which may be composed of substatements defined in Table 1 (e.g. lists, leafs, containers, choices). A new complex type may extend an existing complex type. This allows providing type-safe extensions to existing YANG models as instances of the new type.

Complex types have the following characteristics:

- o Introduction of new types, as a named, formal description of a concrete manageable resource as well as abstract concepts.
- o Types can be extended, i.e. new types can be defined by specializing existing types adding new features. Instances of such an extended type can be used wherever instances of the base type may appear.
- o The type information is made part of the NETCONF payload in case a derived type substitutes a base type. This enables easy and efficient consumption of payload elements representing complex type instances.

2.2. complex-type extension statement

The extension statement "complex-type" is introduced that accepts an arbitrary number of node tree defining statements among other common YANG statements ("YANG Statements", [RFC6020] Section 7).

| substatement | cardinality |
|------------------|-------------|
| abstract | 0..1 |
| anyxml | 0..n |
| choice | 0..n |
| container | 0..n |
| description | 0..1 |
| ct:instance | 0..n |
| ct:instance-list | 0..n |
| ct:extends | 0..1 |
| grouping | 0..n |
| if-feature | 0..n |
| key | 0..1 |
| leaf | 0..n |
| leaf-list | 0..n |
| list | 0..n |
| must | 0..n |
| ordered-by | 0..n |
| reference | 0..1 |
| refine | 0..n |
| status | 0..1 |
| typedef | 0..n |
| uses | 0..n |

Table 1: complex-type's substatements

Complex type definitions may appear at every place, where a grouping may be defined. That includes the module, submodule, rpc, input, output, notification, container, and list statements.

Complex type names populate a distinct namespace. As with YANG groupings, it is possible to define a complex type and a data node (e.g. leaf, list, instance statements) with the same name in the same scope. All complex type names defined within a parent node or at the top-level of the module or its submodules share the same type identifier namespace. This namespace is scoped to the parent node or module.

A complex type MAY have an instance key. An instance key is either defined with the "key" statement as part of the complex type or is inherited from the base complex type. It is not allowed to define an additional key if the base complex type or one of its ancestors already defines a key.

Complex-type definitions do not create nodes in the schema tree.

2.3. instance extension statement

The "instance" extension statement is used to instantiate a complex type by creating a subtree in the management information node tree. The instance statement takes one argument that is the identifier of the complex type instance. It is followed by a block of substatements.

The type of the instance is specified with the mandatory "ct:instance-type" substatement. The type of an instance **MUST** be a complex type. Common YANG statements may be used as substatements of the "instance" statement. An instance is by default optional. To make an instance mandatory, "mandatory true" has to be applied as substatement.

| substatement | cardinality |
|------------------|-------------|
| description | 0..1 |
| config | 0..1 |
| ct:instance-type | 1 |
| if-feature | 0..n |
| mandatory | 0..1 |
| must | 0..n |
| reference | 0..1 |
| status | 0..1 |
| when | 0..1 |
| anyxml | 0..n |
| choice | 0..n |
| container | 0..n |
| ct:instance | 0..n |
| ct:instance-list | 0..n |
| leaf | 0..n |
| leaf-list | 0..n |
| list | 0..n |

Table 2: instance's substatements

The "instance" and "instance-list" extension statements (see Section 2.4 "instance-list extension statement") are similar to the existing "leaf" and "leaf-list" statements, with the exception that the content is composed of subordinate elements according to the instantiated complex type.

It is also possible to add additional data nodes by using the according leaf, leaf-list, list, and choice statements etc. as substatements of the instance declaration. This is an in-place

augmentation of the used complex type confined to a complex type instantiation (see also Section 2.13 "Using complex types" for details on augmenting complex types).

2.4. instance-list extension statement

The "instance-list" extension statement is used to instantiate a complex type by defining a sequence of subtrees in the management information node tree. In addition, the "instance-list" statement takes one argument that is the identifier of the complex type instances. It is followed by a block of substatements.

The type of the instance is specified with the mandatory "ct:instance-type" substatement. In addition it can be defined how often an instance may appear in the schema tree by using the min-elements and max-elements substatements. Common YANG statements may be used as substatements of the "instance-list" statement.

In analogy to "instance" statement, sub-statement like "list", "choice", leaf" etc. MAY be used to augment the instance list elements at the root level with additional data nodes.

| substatement | cardinality |
|------------------|-------------|
| description | 0..1 |
| config | 0..1 |
| ct:instance-type | 1 |
| if-feature | 0..n |
| max-elements | 0..1 |
| min-elements | 0..1 |
| must | 0..n |
| ordered-by | 0..1 |
| reference | 0..1 |
| status | 0..1 |
| when | 0..1 |
| anyxml | 0..n |
| choice | 0..n |
| container | 0..n |
| ct:instance | 0..n |
| ct:instance-list | 0..n |
| leaf | 0..n |
| leaf-list | 0..n |
| list | 0..n |

Table 3: instance-list's substatements

In case the instance list represents configuration data, the used complex type of an instance MUST have an instance key.

Instances as well as instance lists may appear as arguments of the "deviate" statement.

2.5. extends extension statement

A complex type MAY extend exactly one existing base complex type by using the "extends" extension statement. The keyword "extends" MAY occur as substatement of the "complex-type" extension statement. The argument of the "complex-type" extension statement refers to the base complex type via its name. In case a complex type represents configuration data (the default), it MUST have a key, otherwise it MAY have a key. A key is either defined with the key statement as part of the complex type or is inherited from the base complex type.

| substatement | cardinality |
|--------------|-------------|
| description | 0..1 |
| reference | 0..1 |
| status | 0..1 |

Table 4: extends' substatements

2.6. abstract extension statement

Complex types may be declared to be abstract by using the "abstract" extension statement. An abstract complex type cannot be instantiated, meaning it cannot appear as most specific type of an instance in NETCONF payload. In case an abstract type extends a base type, the base complex type MUST be also abstract. By default, complex types are not abstract.

The abstract complex type serves only as a base type for derived concrete complex types and cannot be used as a type for an instance in NETCONF payload.

The "abstract" extension statement takes a single string argument, which is either "true" or "false". In case a "complex-type" statement does not contain an "abstract" statement as substatement, the default is "false". The "abstract" statement does not support any substatements.

2.7. XML Encoding Rules

An "instance" node is encoded as an XML element, where an "instance-list" node is encoded as a series of XML elements. The XML element name is the "instance" respectively "instance-list" identifier, and its XML namespace is the module's XML namespace.

Instance child nodes are encoded as subelements of the instance XML element. Subelements representing child nodes defined in the same complex type may appear in any order. However child nodes of an extending complex type follow the child nodes of the extended complex type. As such, the XML encoding of lists is similar to the encoding of containers and lists in YANG.

Instance key nodes are encoded as subelements of the instance XML element. Instance key nodes must appear in the same order as they are defined within the "key" statement of the according complex type definition and precede all other nodes defined in the same complex type. I.e. if key nodes are defined in an extending complex type, XML elements representing key data precede all other XML elements representing child nodes. On the other hand XML elements representing key data follow the XML elements representing data nodes of the base type.

The type of actual complex type instance is encoded in a type element, which is put in front of all instance child elements, including key nodes, as described in Section 2.8 ("Type Encoding Rules").

The proposed XML encoding rules conform to the YANG XML encoding rules in [RFC6020]. Compared to YANG, enabling key definitions in derived hierarchies is a new feature introduced with the complex types extension. As a new language feature complex types introduce also a new payload entry for the instance type identifier.

Based on our implementation experience, the proposed XML encoding rules support consistent mapping of YANG models with complex types to XML Schema using XML complex types.

2.8. Type Encoding Rules

In order to encode the type of an instance in NETCONF payload, XML elements named "type" belonging to the XML namespace "urn:ietf:params:xml:ns:yang:ietf-complex-type-instance" are added to the serialized form of instance and instance-list nodes in the payload. The suggested namespace prefix is "cti". The "cti:type" XML elements are inserted before the serialized form of all members that have been declared in the according complex type definition.

The "cti:type" element is inserted for each type in the extension chain to the actual type of the instance (most specific last). Each type name includes its corresponding namespace.

The type of a complex type instance MUST be encoded in the reply to NETCONF <get> and <get-config> operations, and in the payload of NETCONF <edit-config> operation if the operation is "create" or "replace". The type of the instance MUST also be specified in case <copy-config> is used to export a configuration to a resource addressed with an URI. The type of the instance has to be specified in user defined RPC's.

The type of the instance MAY be specified in case the operation is "merge" (either because this is explicitly specified or no operation attribute is provided).

In case the node already exists in the target configuration and the type attribute (type of a complex type instance) is specified but differs from the data in the target, an <rpc-error> element is returned with an <error-app-tag> value of "wrong-complex-type". In case no such element is present in the target configuration but the type attribute is missing in the configuration data, an <rpc-error> element is returned with an <error-tag> value of "missing-attribute".

The type MUST NOT be specified in case the operation is "delete".

2.9. Extension and Feature Definition Module

The module below contains all YANG extension definitions for complex types and typed instance identifiers. In addition a "complex-type" feature is defined, which may be used to provide conditional or alternative modeling for depending on the support status of complex types in a NETCONF server. A NETCONF server that supports the complex types modeling features and the XML encoding for complex types as defined in this document MUST advertise this as a feature. This is done by including the feature name "complex-types" into the feature parameter list as part of the NETCONF <hello> message as described in Section 5.6.4 in [RFC6020].

```
<CODE BEGINS> file "ietf-complex-types@2010-10-05.yang"
```

```
module ietf-complex-types {  
  
    namespace "urn:ietf:params:xml:ns:yang:ietf-complex-types";  
    prefix "ct";
```

```
organization
  "NETMOD WG";
```

```
contact
  "Editor:  Bernd Linowski
            <bernd.linowski@ext.nsn.com>
   Editor:  Mehmet Ersue
            <mehmet.ersue@nsn.com>
   Editor:  Siarhei Kuryla
            <s.kuryla@jacobs-university.de>";
```

```
description
  "YANG extensions for complex types and typed instance
  identifiers.
```

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

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

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
// RFC Ed.: Please replace XXXX with actual RFC number and
// remove this note
```

```
revision 2010-10-19 {
  description "Initial revision.";
}
```

```
// RFC Ed.: Please replace the date of the revision statement
// with RFC publication date and remove this note
```

```
extension complex-type {
  description "Defines a complex-type.";
  reference "section 2.2., complex-type extension statement";
  argument type-identifier {
    yin-element true;
  }
}
```

```
extension extends {
    description "Defines the base type of a complex-type.";
    reference "section 2.5., extends extension statement";
    argument base-type-identifier {
        yin-element true;
    }
}

extension abstract {
    description "Makes the complex-type abstract.";
    reference "section 2.6., abstract extension statement";
    argument status;
}

extension instance {
    description "Declares an instance of the given
        complex type.";
    reference "section 2.3., instance extension statement";
    argument ct-instance-identifier {
        yin-element true;
    }
}

extension instance-list {
    description "Declares a list of instances of the given
        complex type";
    reference "section 2.4., instance-list extension statement";
    argument ct-instance-identifier {
        yin-element true;
    }
}

extension instance-type {
    description "Tells to which type instance the instance
        identifier refers to.";
    reference "section 3.2., instance-type extension statement";
    argument target-type-identifier {
        yin-element true;
    }
}

feature complex-types {
    description "This feature indicates that the server supports
        complex types and instance identifiers.";
}
```

```
}
```

```
<CODE ENDS>
```

2.10. Example Model for Complex Types

The example model below shows how complex types can be used to represent physical equipment in a vendor independent, abstract way. It reuses the complex types defined in the physical resource model in Section 1.5.1

<CODE BEGINS>

```
module hw {

    namespace "http://example.com/hw";
    prefix "hw";

    import ietf-complex-types {prefix "ct"; }
    import udmcore {prefix "uc"; }

    // Holder types

    ct:complex-type Slot {
        ct:extends uc:EquipmentHolder;
        leaf slotNumber { type uint16; config false; }
        // ...
    }

    ct:complex-type Chassis {
        ct:extends uc:EquipmentHolder;
        leaf numberOfChassisSlots { type uint32; config false; }
        // ..
    }

    // Equipment types

    ct:complex-type Card {
        ct:extends uc:Equipment;
        leaf position { type uint32; mandatory true; }
        leaf slotsRequired {type uint32; }
    }

    // Root Element
    ct:instance hardware { type uc:ManagedHardware; }

} // hw module

<CODE ENDS>
```

2.11. NETCONF Payload Example

Following example shows the payload of a reply to a NETCONF <get> command. The actual type of managed hardware instances is indicated with the "cti:type" elements as required by the type encoding rules. The containment hierarchy in the NETCONF XML payload reflects the containment hierarchy of hardware instances. This makes filtering

based on the containment hierarchy possible without having to deal with values of key-ref leafs that represent the tree structure in a flattened hierarchy.

```

<hardware>
  <cti:type>uc:BasicObject</cti:type>
  <distinguishedName>/R-T31/CH-2</distinguishedName>
  <globalId>6278279001</globalId>
  <cti:type>uc:Resource</cti:type>
  <cti:type>uc:PhysicalResource</cti:type>
  <otherIdentifier>Rack R322-1</otherIdentifier>
  <serialNumber>R-US-3276279a</serialNumber>
  <cti:type>uc:Hardware</cti:type>
  <cti:type>uc:ManagedHardware</cti:type>
  <cti:type>hw:EquipmentHolder</cti:type>
  <equipmentHolder>
    <cti:type>uc:BasicObject</cti:type>
    <distinguishedName>/R-T31/CH-2/SL-1</distinguishedName>
    <globalId>548872003</globalId>
    <cti:type>uc:Resource</cti:type>
    <cti:type>uc:PhysicalResource</cti:type>
    <otherIdentifier>CU-Slot</otherIdentifier>
    <serialNumber>T-K4733890x45</serialNumber>
    <cti:type>uc:Hardware</cti:type>
    <cti:type>uc:ManagedHardware</cti:type>
    <cti:type>uc:EquipmentHolder</cti:type>
    <equipment>
      <cti:type>uc:BasicObject</cti:type>
      <distinguishedName>/R-T31/CH-2/SL-1/C-3</distinguishedName>
      <globalId>89772001</globalId>
      <cti:type>uc:Resource</cti:type>
      <cti:type>uc:PhysicalResource</cti:type>
      <otherIdentifier>ATM-45252</otherIdentifier>
      <serialNumber>A-778911-b</serialNumber>
      <cti:type>uc:Hardware</cti:type>
      <cti:type>uc:ManagedHardware</cti:type>
      <cti:type>uc:Equipment</cti:type>
      <installed>true</installed>
      <version>A2</version>
      <redundancy>1</redundancy>
      <cti:type>hw:Card</cti:type>
      <usedSlots>1</usedSlots>
    </equipment>
    <cti:type>hw:Slot</cti:type>
    <slotNumber>1</slotNumber>
  </equitmentHolder>
  <cti:type>hw:Chassis</cti:type>
  <numberOfChassisSlots>6</numberOfChassisSlots>
  // ...
</hardware>

```

2.12. Update Rules for Modules Using Complex Types

In addition to the module update rules specified in Section 10 in [RFC6020], modules that define complex-types, instances of complex types and typed instance identifiers must obey following rules:

- o New complex types MAY be added.
- o A new complex type MAY extend an existing complex type.
- o New data definition statements MAY be added to a complex type only if:
 - * they are not mandatory or
 - * they are not conditionally dependent on a new feature (i.e., have an "if-feature" statement, which refers to a new feature).
- o The type referred to by the instance-type statement may be changed to a type that derives from the original type only if the original type does not represent configuration data.

2.13. Using Complex Types

All data nodes defined inside a complex type reside in the complex type namespace, which is their parent node namespace.

2.13.1. Overriding Complex Type Data Nodes

It is not allowed to override a data node inherited from a base type. I.e. it is an error if a type "base" with a leaf named "foo" is extended by another complex type ("derived") with a leaf named "foo" in the same module. In case they are derived in different modules, there are two distinct "foo" nodes which are mapped to the XML namespaces of the module, where the complex types are specified.

A complex type that extends a basic complex type may use the "refine" statement in order to improve an inherited data node. The target node identifier must be qualified by the module prefix to indicate clearly, which inherited node is refined.

The following refinements can be done:

- o A leaf or choice node may have a default value, or a new default value if it already had one
- o Any node may have a different "description" or "reference" string.

- o A leaf, anyxml, or choice node may have a "mandatory true" statement. However, it is not allowed to change from "mandatory true" to "mandatory false".
- o A leaf, leaf-list, list, container, or anyxml node may have additional "must" expressions.
- o A list, leaf-list, instance or instance-list node may have a "min-elements" statement, if the base type does not have one or one with a value that is greater than the minimum value of the base type.
- o A list, leaf-list, instance or instance-list node may have a "max-elements" statement, if the base type does not have one or one with a value that is smaller than the maximum value of the base type.

It is not allowed to refine complex-type nodes inside instance or instance-list statements.

2.13.2. Augmenting Complex Types

Augmenting complex types is only allowed if a complex type is instantiated in an "instance" or "instance-list" statement. This confines the effect of the augmentation to the location in the schema tree, where the augmentation is done. The argument of the "augment" statement MUST be in the descendant form (as defined by the rule "descendant-schema-nodeid" in Section 12 in [RFC6020]).

```
ct:complex-type Chassis {
    ct:extends EquipmentHolder;
    container chassisInfo {
        config false;
        leaf numberOfSlots { type uint16; }
        leaf occupiedSlots { type uint16; }
        leaf height {type unit16;}
        leaf width {type unit16;}
    }
}

ct:instance-list chassis {
    type Chassis;
    augment "chassisInfo" {
        leaf modelId { type string; }
    }
}
```

When augmenting a complex type, only the "container", "leaf", "list", "leaf-list", "choice", "instance", "instance-list" and "if-feature" statements may be used within the "augment" statement. The nodes added by the augmentation MUST NOT be mandatory nodes. One or many augment statements may not cause the creation of multiple nodes with the same name from the same namespace in the target node.

To achieve less complex modeling this document proposes the augmentation of complex type instances without recursion.

2.13.3. Controlling the Use of Complex Types

A server might not want to support all complex types defined in a supported module. This issue can be addressed with YANG features as follows:

- o Features are defined that are used inside complex type definitions (by using "if-feature" as substatement) to make them optional. In this case such complex types may only be instantiated if the feature is supported (advertized as capability in the NETCONF <hello> message).
- o The "deviation" statement may be applied to node trees, which are created by "instance" and "instance-list" statements. In this case, only the substatement "deviate not-supported" is allowed.
- o It is not allowed to apply the deviation statement to node tree elements that may occur because of the recursive use of a complex type. Other forms of deviations ("deviate add", "deviate replace", "deviate delete") are NOT supported inside node trees spanned by "instance" or "instance-list".

As complex type definitions do not contribute by itself to the data node tree, data node declarations inside complex types cannot be target of deviations.

In the example below, client applications are informed that the leaf "occupiedSlots" is not supported in the top-level chassis. However, if a chassis contains another chassis, the contained chassis may support the leaf informing about the number of occupied slots.

```
deviation "/chassis/chassisSpec/occupiedSlots" {  
    deviate not-supported;  
}
```

3. Typed Instance Identifier

3.1. Definition

Typed instance identifier relationships are an addition to the relationship types already defined in YANG, where the leafref relationship is location dependent, and the instance-identifier does not specify to which type of instances the identifier points to.

A typed instance identifier represents a reference to an instance of a complex type without being restricted to a particular location in the containment tree. This is done by using the extension statement "instance-type" as a substatement of the existing "type instance identifier" statement.

Typed instance identifiers allow referring to instances of complex types that may be located anywhere in the schema tree. The "type" statement plays the role of a restriction that must be fulfilled by the target node, which is referred to with the instance identifier. The target node **MUST** be of a particular complex type, either the type itself or any type that extends this complex type.

3.2. instance-type extension statement

The "instance-type" extension statement specifies the complex type of the instance referred by the instance-identifier. The referred instance may also instantiate any complex type that extends the specified complex type.

The instance complex type is identified by the single name argument. The referred complex type **MUST** have a key. This extension statement **MUST** be used as a substatement of the "type instance-identifier" statement. The "instance-type" extension statement does not support any substatements.

3.3. Typed Instance Identifier Example

In the example below, a physical link connects an arbitrary number of physical ports. Here typed instance identifiers are used to denote, which "PhysicalPort" instances (anywhere in the data tree) are connected by a "PhysicalLink".

```
// Extended version of type Card
ct:complex-type Card {
  ct:extends Equipment;
  leaf usedSlot { type uint16; mandatory true; }
  ct:instance-list port {
    type PhysicalPort;
  }
}

ct:complex-type PhysicalPort {
  ct:extends ManagedHardware;
  leaf portNumber { type int32; mandatory true; }
}

ct:complex-type PhysicalLink {
  ct:extends ManagedHardware;
  leaf media { type string; }
  leaf-list connectedPort {
    type instance-identifier {
      ct:instance-type PhysicalPort;
    }
    min-elements 2;
  }
}
```

Below is the XML encoding of an element named "link" of type "PhysicalLink":

```
<link>
  <objectId>FTCL-771</objectId>
  <media>Fiber</media>
  <connectedPort>/hw:hardware[objectId='R-11']
    /hw:equipment[objectId='AT22']/hw:port[objectId='P12']
  </connectedPort>
  <connectedPort>/hw:hardware[objectId='R-42']
    /hw:equipment[objectId='AT30']/hw:port[objectId='P3']
  </connectedPort>
  <serialNumeber>F-7786828</serialNumber>
  <commonName>FibCon 7</commonName>
</link>
```

4. IANA Considerations

This document registers two URIs in the IETF XML registry. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-complex-types
URI: urn:ietf:params:xml:ns:yang:ietf-complex-type-instance

Registrant Contact: The NETMOD WG of the IETF.

XML: N/A, the requested URIs are XML namespaces.

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-complex-types

namespace: urn:ietf:params:xml:ns:yang:ietf-complex-types

prefix: ct

RFC: XXXX

RFC Ed.: Please replace XXXX with actual RFC number and remove this note.

5. Security Considerations

The YANG module "complex-types" in this memo defines YANG extensions for Complex-types and Typed Instance Identifiers as new language statements.

Complex-types and Typed Instance Identifiers themselves do not have any security impact on the Internet.

The security considerations described throughout [RFC6020] apply here as well.

6. Acknowledgements

The authors would like to thank to Martin Bjorklund, Balazs Lengyel, Gerhard Muenz, Dan Romascanu, Juergen Schoenwaelder and Martin Storch for their valuable review and comments on different versions of the document.

7. References

7.1. Normative References

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

- [RFC3688] Mealling, M., "The IETF XML Registry", January 2004.
- [RFC5226] Narten, T., "Guidelines for Writing an IANA Considerations Section in RFCs", May 2008.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", October 2010.

7.2. Informative References

- [IPFIXCONF] Muenz, G., "Configuration Data Model for IPFIX and PSAMP", draft-ietf-ipfix-configuration-model-07 (work in progress), July 2010.
- [Libsmi] Kuryla, S., "Libsmi Extension for Complex Types", April 2010, <<http://www.ibr.cs.tu-bs.de/svn/libsmi>>.
- [Pyang] Bjorklund, M., "An extensible YANG validator and converter", October 2010, <<http://code.google.com/p/pyang/>>.
- [Pyang-ct] Kuryla, S., "Pyang Extension for Complex Types", April 2010, <<http://code.google.com/p/pyang-ct/>>.
- [RFC4133] Bierman, A. and K. McCloghrie, "Entity MIB (Version 3)", August 2005.
- [SID V8] Tele Management Forum, "GB922-Information Framework (SID) Solution Suite, Release 8.0", July 2008, <<http://www.tmforum.org/DocumentsInformation/GB922InformationFramework/35499/article.html>>.
- [UDM] NSN, "Unified Data Model SID Compliance Statement", May 2010, <<http://www.tmforum.org/InformationFramework/NokiaSiemensNetworks/8815/home.html>>.

Appendix A. Change Log

A.1. 03-04

- o Changed the complex type XML encoding rules so that XML elements representing data nodes defined in the same complex type may appear in any order.
- o Used the "ct:" prefix in substatement tables when referring to complex type extension statements.

- o Modeled the IPFIX/PSMAP example based on v-07 of the IPFIX configuration draft. Changed motivation text accordingly.
- o Minor updates and clarifications in the text.

A.2. 02-03

- o Added an example based on the physical resource modeling concepts of SID. A simplified class diagram and an excerpt of an according YANG module were added in the introduction section.
- o Changed the example YANG module in the NETCONF payload section to be based on the physical resource types defined in the added physical resource model.
- o A second example shows how Entity MIB entries can be modeled as physical resources. The example includes a class diagram and an according YANG module excerpt.
- o The complete YANG modules for both examples were added into the appendix.
- o Changed the complex type encoding rules.
- o Updated the NETCONF payload example the changed type encoding rules and the changed example module.
- o Changed the augmentation rules for complex types. Instead of using "." as argument in the augment statement, instance and instance-list statement may now contain additional data node statements. The substatement tables for the instance and instance-list statements were updated accordingly.
- o Minor updates in the text and examples.

A.3. 01-02

- o It is no longer allowed to use the "config" statement inside a complex type definition.
- o Complex type can now be defined where a grouping can be defined. Complex types have their own namespace.
- o Explicitly specified which kind of refinements can be applied to elements of the base type in the definition of an extending complex type.

- o Confined the use of deviations for complex types to complex type instantiations.
- o Defined augmentation of complex types allowing augmentation only during instantiation via an "instance" or "instance-list" statement.
- o Removed leftovers from substatement tables.
- o Updates and bug-fixes in the examples.

A.4. 00-01

- o Transformed proposed new YANG statements to YANG extension statements (complex-type, element, extends, abstract).
- o Renamed statement "element" to the extension statement "instance" in order to avoid confusion with XML payload elements.
- o Introduced extension statement "instance-type" as allowing the use of the existing "type" statement as substatement in the existing "instance-identifier" statement cannot be done with extensions.
- o Added the complex type extension statement module.
- o Updated examples to reflect the changes mentioned above.
- o Added update rules for complex types.
- o Updated IANA Considerations section.
- o Added this change log.

Appendix B. YANG Modules for Physical Network Resource Model and Hardware Entities Model

YANG module for the 'Physical Network Resource Model':

<CODE BEGINS>

```
module udmcore {  
  
    namespace "http://example.com/udmcore";  
    prefix "udm";  
  
    import ietf-yang-types {prefix "yang";}   
    import ietf-complex-types {prefix "ct";}   
}
```

```
ct:complex-type BasicObject {
  ct:abstract true;
  key "distinguishedName";
  leaf globalId {type int64;}
  leaf distinguishedName {type string; mandatory true;}
}

ct:complex-type ManagedObject {
  ct:extends BasicObject;
  ct:abstract true;
  leaf instance {type string;}
  leaf objectState {type int32;}
  leaf release {type string;}
}

ct:complex-type Resource {
  ct:extends ManagedObject;
  ct:abstract true;
  leaf usageState {type int16;}
  leaf managementMethodSupported {type string;}
  leaf managementMethodCurrent {type string;}
  leaf managementInfo {type string;}
  leaf managementDomain {type string;}
  leaf version {type string;}
  leaf entityIdentification {type string;}
  leaf description {type string;}
  leaf rootEntityType {type string;}
}

ct:complex-type LogicalResource {
  ct:extends Resource;
  ct:abstract true;
  leaf lrStatus {type int32;}
  leaf serviceState {type int32;}
  leaf isOperational {type boolean;}
}

ct:complex-type PhysicalResource {
  ct:extends Resource;
  ct:abstract true;
  leaf manufactureDate {type string;}
  leaf otherIdentifier {type string;}
  leaf powerState {type int32;}
  leaf serialNumber {type string;}
  leaf versionNumber {type string;}
}
```

```
}

ct:complex-type Hardware {
  ct:extends PhysicalResource;
  ct:abstract true;
  leaf width {type string;}
  leaf height {type string;}
  leaf depth {type string;}
  leaf measurementUnits {type int32;}
  leaf weight {type string;}
  leaf weightUnits {type int32;}
  leaf-list physicalLink {
    type instance-identifier {
      ct:instance-type PhysicalLink;
    }
  }
  ct:instance-list containedHardware {
    ct:instance-type Hardware;
  }
  ct:instance-list physicalConnector {
    ct:instance-type PhysicalConnector;
  }
}

ct:complex-type PhysicalLink {
  ct:extends PhysicalResource;
  leaf isWireless {type boolean;}
  leaf currentLength {type string;}
  leaf maximumLength {type string;}
  leaf mediaType {type int32;}
  leaf-list hardware {
    type instance-identifier {
      ct:instance-type Hardware;
    }
  }
}

ct:complex-type ManagedHardware {
  ct:extends Hardware;
  leaf additionalInfo {type string;}
  leaf physicalAlarmReportingEnabled {type boolean;}
  leaf physicalAlarmStatus {type int32;}
  leaf coolingRequirements {type string;}
  leaf hardwarePurpose {type string;}
  leaf isPhysicalContainer {type boolean;}
```

```
    }

    ct:complex-type AuxiliaryComponent {
        ct:extends ManagedHardware;
        ct:abstract true;
    }

    ct:complex-type PhysicalPort {
        ct:extends ManagedHardware;
        leaf portNumber {type int32;}
        leaf duplexMode {type int32;}
        leaf ifType {type int32;}
        leaf vendorPortName {type string;}
    }

    ct:complex-type PhysicalConnector {
        ct:extends Hardware;
        leaf location {type string;}
        leaf cableType {type int32;}
        leaf gender {type int32;}
        leaf inUse {type boolean;}
        leaf pinDescription {type string;}
        leaf typeOfConnector {type int32;}
        leaf-list sourcePhysicalConnector {
            type instance-identifier {
                ct:instance-type PhysicalConnector;
            }
        }
        leaf-list targetPhysicalConnector {
            type instance-identifier {
                ct:instance-type PhysicalConnector;
            }
        }
    }

    ct:complex-type Equipment {
        ct:extends ManagedHardware;
        leaf installStatus {type int32;}
        leaf expectedEquipmentType {type string;}
        leaf installedEquipmentType {type string;}
        leaf installedVersion {type string;}
        leaf redundancy {type int32;}
        leaf vendorName {type string;}
        leaf dateOfLastService {type yang:date-and-time;}
```

```
    leaf interchangeability {type string;}
    leaf identificationCode {type string;}
    ct:instance-list equipment {
        ct:instance-type Equipment;
    }
}

ct:complex-type EquipmentHolder {
    ct:extends ManagedHardware;
    leaf vendorName {type string;}
    leaf locationName {type string;}
    leaf dateOfLastService {type yang:date-and-time;}
    leaf partNumber {type string;}
    leaf availabilityStatus {type int16;}
    leaf nameFromPlanningSystem {type string;}
    leaf modelNumber {type string;}
    leaf acceptableEquipmentList {type string;}
    leaf isSolitaryHolder {type boolean;}
    leaf holderStatus {type int16;}
    leaf interchangeability {type string;}
    leaf equipmentHolderSpecificType {type string;}
    leaf position {type string;}
    leaf atomicCompositeType {type int16;}
    leaf uniquePhysical {type boolean;}
    leaf physicalDescription {type string;}
    leaf serviceApproach {type string;}
    leaf mountingOptions {type int32;}
    leaf cableManagementStrategy {type string;}
    leaf isSecureHolder {type boolean;}
    ct:instance-list equipment {
        ct:instance-type Equipment;
    }
    ct:instance-list equipmentHolder {
        ct:instance-type EquipmentHolder;
    }
}

// ... other resource complex types ...
}
<CODE ENDS>
```

YANG module for the 'Hardware Entities Model':

<CODE BEGINS>

```
module hardware-entities {

    namespace "http://example.com/:hardware-entities";
    prefix "hwe";

    import ietf-yang-types {prefix "yang";}
    import ietf-complex-types {prefix "ct";}
    import udmcore {prefix "uc";}

    grouping PhysicalEntityProperties {
        leaf hardwareRev {type string; }
        leaf firmwareRev {type string; }
        leaf softwareRev {type string; }
        leaf serialNum {type string; }

        leaf mfgName {type string; }
        leaf modelName {type string; }
        leaf alias {type string; }
        leaf ssetID {type string; }
        leaf isFRU {type boolean; }
        leaf mfgDate {type yang:date-and-time; }
        leaf-list uris {type string; }
    }

    // Physical entities representing equipment

    ct:complex-type Module {
        ct:extends uc:Equipment;
        description "Complex type representing module entries
            (entPhysicalClass = module(9)) in entPhysicalTable";
        uses PhysicalEntityProperties;
    }

    ct:complex-type Backplane {
        ct:extends uc:Equipment;
        description "Complex type representing backplane entries
            (entPhysicalClass = backplane(4)) in entPhysicalTable";
        uses PhysicalEntityProperties;
    }

    // Physical entities representing auxiliary hardware components

    ct:complex-type PowerSupply {
        ct:extends uc:AuxiliaryComponent;
        description "Complex type representing power supply entries
```



```
        (entPhysicalClass = powerSupply(6)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

ct:complex-type Fan {
    ct:extends uc:AuxiliaryComponent;
    description "Complex type representing fan entries
        (entPhysicalClass = fan(7)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

ct:complex-type Sensor {
    ct:extends uc:AuxiliaryComponent;
    description "Complex type representing sensor entries
        (entPhysicalClass = sensor(8)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

// Physical entities representing equipment holders

ct:complex-type Chassis {
    ct:extends uc:EquipmentHolder;
    description "Complex type representing chassis entries
        (entPhysicalClass = chassis(3)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

ct:complex-type Container {
    ct:extends uc:EquipmentHolder;
    description "Complex type representing container entries
        (entPhysicalClass = container(5)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

ct:complex-type Stack {
    ct:extends uc:EquipmentHolder;
    description "Complex type representing stack entries
        (entPhysicalClass = stack(11)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

// Other kinds of physical entities

ct:complex-type Port {
    ct:extends uc:PhysicalPort;
    description "Complex type representing port entries
```

```
        (entPhysicalClass = port(10)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

ct:complex-type CPU {
    ct:extends uc:Hardware;
    description "Complex type representing cpu entries
        (entPhysicalClass = cpu(12)) in entPhysicalTable";
    uses PhysicalEntityProperties;
}

}
<CODE ENDS>
```

Appendix C. Example YANG Module for the IPFIX/PSAMP Model

C.1. Modeling Improvements for the IPFIX/PSAMP Model with Complex types and Typed instance identifiers

The module below is a variation of the IPFIX/PSAMP configuration model, which uses complex types and typed instance identifiers to model the concept outlined in [IPFIXCONF].

When looking at the YANG module with complex types and typed instance identifiers, various technical improvements on modeling level become apparent.

- o There is almost a one-to-one mapping between the domain concepts introduced in IPFIX and the complex types in the YANG module
- o All associations between the concepts (which are not containment) are represented with typed identifiers. That avoids having to refer to a particular location in the tree, which is not mandated by the original model.
- o It is superfluous to represent concept refinement (class inheritance in the original model) with containment in form of quite big choice-statements with complex branches. Instead, concept refinement is realized by complex types extending a base complex type.
- o It is unnecessary to introduce metadata identities and leafs (e.g. "identity cacheMode" and "leaf cacheMode" in "grouping cacheParameters") that just serve the purpose of indicating which concrete sub-type of a generic type (modeled as grouping, which contains the union of all features of all subtypes) is actually represented in the MIB.

- o Ruling out illegal use of sub-type specific properties (e.g. "leaf maxFlows") by using "when" statements that refer to a sub-type discriminator is not necessary (e.g. when "../cacheMode != 'immediate'").
- o It is not needed to define properties like the configuration status wherever a so called "parameter grouping" is used. Instead those definitions can be put inside the complex-type definition itself.
- o It can be avoided to separating the declaration of the key from the related data nodes definitions in a grouping (see use of "grouping selectorParameters").
- o Complex types may be declared as optional features. If the type is indicated with an identity (e.g. "identity immediate"), this is not possible, since "if-feature" is not allowed as a substatement of "identity".

C.2. IPFIX/PSAMP Model with Complex Types and Typed Instance Identifiers

<CODE BEGINS>

```
module ct-ipfix-psamp-example {
  namespace "http://example.com/ns/ct-ipfix-psamp-example";
  prefix ipfix;

  import ietf-yang-types { prefix yang; }
  import ietf-inet-types { prefix inet; }
  import ietf-complex-types {prefix "ct"; }

  description "Example IPFIX/PSAMP Configuration Data Model
    with complex types and typed instance identifiers";

  revision 2010-10-19 {
    description "Version of draft-ietf-ipfix-configuration-model-07
      modeled with complex types and typed instance identifiers.";
  }

  /*****
  * Features
  *****/

  feature exporter {
    description "If supported, the Monitoring Device can be used as
      an Exporter. Exporting Processes can be configured.";
  }
```

```
feature collector {
  description "If supported, the Monitoring Device can be used as
    a Collector. Collecting Processes can be configured.";
}

feature meter {
  description "If supported, Observation Points, Selection
    Processes, and Caches can be configured.";
}

feature psampSampCountBased {
  description "If supported, the Monitoring Device supports
    count-based Sampling...";
}

feature psampSampTimeBased {
  description "If supported, the Monitoring Device supports
    time-based Sampling...";
}

feature psampSampRandOutOfN {
  description "If supported, the Monitoring Device supports
    random n-out-of-N Sampling...";
}

feature psampSampUniProb {
  description "If supported, the Monitoring Device supports
    uniform probabilistic Sampling...";
}

feature psampFilterMatch {
  description "If supported, the Monitoring Device supports
    property match Filtering...";
}

feature psampFilterHash {
  description "If supported, the Monitoring Device supports
    hash-based Filtering...";
}

feature cacheModeImmediate {
  description "If supported, the Monitoring Device supports
    Cache Mode 'immediate'.";
}

feature cacheModeTimeout {
  description "If supported, the Monitoring Device supports
    Cache Mode 'timeout'.";
```

```
}

feature cacheModeNatural {
  description "If supported, the Monitoring Device supports
    Cache Mode 'natural'.";
}

feature cacheModePermanent {
  description "If supported, the Monitoring Device supports
    Cache Mode 'permanent'.";
}

feature udpTransport {
  description "If supported, the Monitoring Device supports UDP
    as transport protocol.";
}

feature tcpTransport {
  description "If supported, the Monitoring Device supports TCP
    as transport protocol.";
}

feature fileReader {
  description "If supported, the Monitoring Device supports the
    configuration of Collecting Processes as File Readers.";
}

feature fileWriter {
  description "If supported, the Monitoring Device supports the
    configuration of Exporting Processes as File Writers.";
}

/*****
* Identities
*****/

/** Hash function identities */
identity hashFunction {
  description "Base identity for all hash functions...";
}
identity BOB {
  base "hashFunction";
  description "BOB hash function";
  reference "RFC5475, Section 6.2.4.1.";
}
identity IPSX {
  base "hashFunction";
  description "IPSX hash function";
```

```
    reference "RFC5475, Section 6.2.4.1.";
}
identity CRC {
    base "hashFunction";
    description "CRC hash function";
    reference "RFC5475, Section 6.2.4.1.";
}

/** Export mode identities */
identity exportMode {
    description "Base identity for different usages of export
        destinations configured for an Exporting Process...";
}
identity parallel {
    base "exportMode";
    description "Parallel export of Data Records to all
        destinations configured for the Exporting Process.";
}
identity loadBalancing {
    base "exportMode";
    description "Load-balancing between the different
        destinations...";
}
identity fallback {
    base "exportMode";
    description "Export to the primary destination...";
}

/** Options type identities */
identity optionsType {
    description "Base identity for report types exported
        with options...";
}
identity meteringStatistics {
    base "optionsType";
    description "Metering Process Statistics.";
    reference "RFC 5101, Section 4.1.";
}
identity meteringReliability {
    base "optionsType";
    description "Metering Process Reliability Statistics.";
    reference "RFC 5101, Section 4.2.";
}
identity exportingReliability {
    base "optionsType";
    description "Exporting Process Reliability
        Statistics.";
    reference "RFC 5101, Section 4.3.";
```

```
}
identity flowKeys {
  base "optionsType";
  description "Flow Keys.";
  reference "RFC 5101, Section 4.4.";
}
identity selectionSequence {
  base "optionsType";
  description "Selection Sequence and Selector Reports.";
  reference "RFC5476, Sections 6.5.1 and 6.5.2.";
}
identity selectionStatistics {
  base "optionsType";
  description "Selection Sequence Statistics Report.";
  reference "RFC5476, Sections 6.5.3.";
}
identity accuracy {
  base "optionsType";
  description "Accuracy Report.";
  reference "RFC5476, Section 6.5.4.";
}
identity reducingRedundancy {
  base "optionsType";
  description "Enables the utilization of Options Templates to
    reduce redundancy in the exported Data Records.";
  reference "RFC5473.";
}
identity extendedTypeInfo {
  base "optionsType";
  description "Export of extended type information for
    enterprise-specific Information Elements used in the
    exported Templates.";
  reference "RFC5610.";
}

/*****
* Type definitions
*****/

typedef nameType {
  type string {
    length "1..max";
    pattern "\\S(\\.\\S)?";
  }
  description "Type for 'name' leafs...";
}

typedef direction {
```

```
type enumeration {
  enum ingress {
    description "This value is used for monitoring incoming
      packets.";
  }
  enum egress {
    description "This value is used for monitoring outgoing
      packets.";
  }
  enum both {
    description "This value is used for monitoring incoming and
      outgoing packets.";
  }
}
description "Direction of packets going through an interface or
  linecard.";
}

typedef transportSessionStatus {
  type enumeration {
    enum inactive {
      description "This value MUST be used for...";
    }
    enum active {
      description "This value MUST be used for...";
    }
    enum unknown {
      description "This value MUST be used if the status...";
    }
  }
  description "Status of a Transport Session.";
  reference "RFC5815, Section 8 (ipfixTransportSessionStatus).";
}

/*****
* Complex types
*****/

ct:complex-type ObservationPoint {
  description "Observation Point";
  key name;
  leaf name {
    type nameType;
    description "Key of an observation point.";
  }
  leaf observationPointId {
    type uint32;
    config false;
  }
}
```



```
    description "Observation Point ID...";
    reference "RFC5102, Section 5.1.10.";
  }
  leaf observationDomainId {
    type uint32;
    mandatory true;
    description "The Observation Domain ID associates...";
    reference "RFC5101.";
  }
  choice OPLocation {
    mandatory true;
    description "Location of the Observation Point.";
    leaf ifIndex {
      type uint32;
      description "Index of an interface...";
      reference "RFC 1229.";
    }
    leaf ifName {
      type string;
      description "Name of an interface...";
      reference "RFC 1229.";
    }
    leaf entPhysicalIndex {
      type uint32;
      description "Index of a linecard...";
      reference "RFC 4133.";
    }
    leaf entPhysicalName {
      type string;
      description "Name of a linecard...";
      reference "RFC 4133.";
    }
  }
  leaf direction {
    type direction;
    default both;
    description "Direction of packets....";
  }
  leaf-list selectionProcess {
    type instance-identifier { ct:instance-type SelectionProcess; }
    description "Selection Processes in this list process packets
      in parallel.";
  }
}

ct:complex-type Selector {
  ct:abstract true;
  description "Abstract selector";
}
```

```
key name;
leaf name {
    type nameType;
    description "Key of a selector";
}
leaf packetsObserved {
    type yang:counter64;
    config false;
    description "The number of packets observed ...";
    reference "RFC5815, Section 8
        (ipfixSelectorStatsPacketsObserved).";
}
leaf packetsDropped {
    type yang:counter64;
    config false;
    description "The total number of packets discarded ...";
    reference "RFC5815, Section 8
        (ipfixSelectorStatsPacketsDropped).";
}
leaf selectorDiscontinuityTime {
    type yang:date-and-time;
    config false;
    description "Timestamp of the most recent occasion at which
        one or more of the Selector counters suffered a
        discontinuity...";
    reference "RFC5815, Section 8
        (ipfixSelectionProcessStatsDiscontinuityTime).";
}
}

ct:complex-type SelectAllSelector {
    ct:extends Selector;
    description "Method which selects all packets.";
}

ct:complex-type SampCountBasedSelector {
    if-feature psampSampCountBased;
    ct:extends Selector;
    description "Selector applying systematic count-based
        packet sampling to the packet stream.";
    reference "RFC5475, Section 5.1;
        RFC5476, Section 6.5.2.1.";
    leaf packetInterval {
        type uint32;
        units packets;
        mandatory true;
        description "The number of packets that are consecutively
            sampled between gaps of length packetSpace.
```

```
        This parameter corresponds to the Information Element
        samplingPacketInterval.";
        reference "RFC5477, Section 8.2.2.";
    }
    leaf packetSpace {
        type uint32;
        units packets;
        mandatory true;
        description "The number of unsampled packets between two
            sampling intervals.
            This parameter corresponds to the Information Element
            samplingPacketSpace.";
        reference "RFC5477, Section 8.2.3.";
    }
}

ct:complex-type SampTimeBasedSelector {
    if-feature psampSampTimeBased;
    ct:extends Selector;
    description "Selector applying systematic time-based
        packet sampling to the packet stream.";
    reference "RFC5475, Section 5.1;
        RFC5476, Section 6.5.2.2.";
    leaf timeInterval {
        type uint32;
        units microseconds;
        mandatory true;
        description "The time interval in microseconds during
            which all arriving packets are sampled between gaps
            of length timeSpace.
            This parameter corresponds to the Information Element
            samplingTimeInterval.";
        reference "RFC5477, Section 8.2.4.";
    }
    leaf timeSpace {
        type uint32;
        units microseconds;
        mandatory true;
        description "The time interval in microseconds during
            which no packets are sampled between two sampling
            intervals specified by timeInterval.
            This parameter corresponds to the Information Element
            samplingTimeInterval.";
        reference "RFC5477, Section 8.2.5.";
    }
}

ct:complex-type SampRandOutOfNSelector {
```

```
if-feature psampSampRandOutOfN;
ct:extends Selector;
description "This container contains the configuration
  parameters of a Selector applying n-out-of-N packet
  sampling to the packet stream.";
reference "RFC5475, Section 5.2.1;
  RFC5476, Section 6.5.2.3.";
leaf size {
  type uint32;
  units packets;
  mandatory true;
  description "The number of elements taken from the parent
    population.
    This parameter corresponds to the Information Element
    samplingSize.";
  reference "RFC5477, Section 8.2.6.";
}
leaf population {
  type uint32;
  units packets;
  mandatory true;
  description "The number of elements in the parent
    population.
    This parameter corresponds to the Information Element
    samplingPopulation.";
  reference "RFC5477, Section 8.2.7.";
}
}

ct:complex-type SampUniProbSelector {
  if-feature psampSampUniProb;
  ct:extends Selector;
  description "Selector applying uniform probabilistic
    packet sampling (with equal probability per packet) to the
    packet stream.";
  reference "RFC5475, Section 5.2.2.1;
    RFC5476, Section 6.5.2.4.";
  leaf probability {
    type decimal64 {
      fraction-digits 18;
      range "0..1";
    }
    mandatory true;
    description "Probability that a packet is sampled,
      expressed as a value between 0 and 1. The probability
      is equal for every packet.
      This parameter corresponds to the Information Element
      samplingProbability.";
```

```
        reference "RFC5477, Section 8.2.8.";
    }
}

ct:complex-type FilterMatchSelector {
    if-feature psampFilterMatch;
    ct:extends Selector;
    description "This container contains the configuration
        parameters of a Selector applying property match filtering
        to the packet stream.";
    reference "RFC5475, Section 6.1;
        RFC5476, Section 6.5.2.5.";
    choice nameOrId {
        mandatory true;
        description "The field to be matched is specified by
            either the name or the ID of the Information
            Element.";
        leaf ieName {
            type string;
            description "Name of the Information Element.";
        }
        leaf ieId {
            type uint16 {
                range "1..32767" {
                    description "Valid range of Information Element
                        identifiers.";
                    reference "RFC5102, Section 4.";
                }
            }
            description "ID of the Information Element.";
        }
    }
    leaf ieEnterpriseNumber {
        type uint32;
        description "If present, ... ";
    }
    leaf value {
        type string;
        mandatory true;
        description "Matching value of the Information Element.";
    }
}

ct:complex-type FilterHashSelector {
    if-feature psampFilterHash;
    ct:extends Selector;
    description "This container contains the configuration
        parameters of a Selector applying hash-based filtering
```

```
    to the packet stream.";
reference "RFC5475, Section 6.2;
RFC5476, Section 6.5.2.6.";
leaf hashFunction {
    type identityref {
        base "hashFunction";
    }
    default BOB;
    description "Hash function to be applied. According to
RFC5475, Section 6.2.4.1, 'BOB' must be used in order to
be compliant with PSAMP.";
}
leaf ipPayloadOffset {
    type uint64;
    units octets;
    default 0;
    description "IP payload offset ... ";
    reference "RFC5477, Section 8.3.2.";
}
leaf ipPayloadSize {
    type uint64;
    units octets;
    default 8;
    description "Number of IP payload bytes ... ";
    reference "RFC5477, Section 8.3.3.";
}
leaf digestOutput {
    type boolean;
    default false;
    description "If true, the output ... ";
    reference "RFC5477, Section 8.3.8.";
}
leaf initializerValue {
    type uint64;
    description "Initializer value to the hash function.
    If not configured by the user, the Monitoring Device
    arbitrarily chooses an initializer value.";
    reference "RFC5477, Section 8.3.9.";
}
list selectedRange {
    key name;
    min-elements 1;
    description "List of hash function return ranges for
    which packets are selected.";
    leaf name {
        type nameType;
        description "Key of this list.";
    }
}
```

```
    leaf min {
      type uint64;
      description "Beginning of the hash function's selected
        range.
        This parameter corresponds to the Information Element
        hashSelectedRangeMin.";
      reference "RFC5477, Section 8.3.6.";
    }
    leaf max {
      type uint64;
      description "End of the hash function's selected range.
        This parameter corresponds to the Information Element
        hashSelectedRangeMax.";
      reference "RFC5477, Section 8.3.7.";
    }
  }
}

ct:complex-type Cache {
  ct:abstract true;
  description "Cache of a Monitoring Device.";
  key name;
  leaf name {
    type nameType;
    description "Key of a cache";
  }
  leaf-list exportingProcess {
    type leafref { path "/ipfix/exportingProcess/name"; }
    description "Records are exported by all Exporting Processes
      in the list.";
  }
  description "Configuration and state parameters of a Cache.";
  container cacheLayout {
    description "Cache Layout.";
    list cacheField {
      key name;
      min-elements 1;
      description "List of fields in the Cache Layout.";
      leaf name {
        type nameType;
        description "Key of this list.";
      }
    }
    choice nameOrId {
      mandatory true;
      description "Name or ID of the Information Element.";
      reference "RFC5102.";
      leaf ieName {
        type string;
      }
    }
  }
}
```

```
        description "Name of the Information Element.";
    }
    leaf ieId {
        type uint16 {
            range "1..32767" {
                description "Valid range of Information Element
                    identifiers.";
                reference "RFC5102, Section 4.";
            }
        }
        description "ID of the Information Element.";
    }
}
leaf ieLength {
    type uint16;
    units octets;
    description "Length of the field ... ";
    reference "RFC5101, Section 6.2; RFC5102.";
}
leaf ieEnterpriseNumber {
    type uint32;
    description "If present, the Information Element is
        enterprise-specific. ... ";
    reference "RFC5101; RFC5102.";
}
leaf isFlowKey {
    when "(../../../../../cacheMode != 'immediate')
        and
        ((count(../ieEnterpriseNumber) = 0)
        or
        (../ieEnterpriseNumber != 29305))" {
        description "This parameter is not available
            for Reverse Information Elements (which have
            enterprise number 29305) or if the Cache Mode
            is 'immediate'.";
    }
    type empty;
    description "If present, this is a flow key.";
}
}
leaf dataRecords {
    type yang:counter64;
    units "Data Records";
    config false;
    description "The number of Data Records generated ... ";
    reference "ietf-draft-ipfix-mib-10, Section 8
        (ipfixMeteringProcessDataRecords).";
}
```



```
    }
    leaf cacheDiscontinuityTime {
        type yang:date-and-time;
        config false;
        description "Timestamp of the ... ";
        reference "ietf-draft-ipfix-mib-10, Section 8
            (ipfixMeteringProcessDiscontinuityTime).";
    }
}

ct:complex-type ImmediateCache {
    if-feature cacheModeImmediate;
    ct:extends Cache;
}

ct:complex-type NonImmediateCache {
    ct:abstract true;
    ct:extends Cache;
    leaf maxFlows {
        type uint32;
        units flows;
        description "This parameter configures the maximum number of
            Flows in the Cache ... ";
    }
    leaf activeFlows {
        type yang:gauge32;
        units flows;
        config false;
        description "The number of Flows currently active in this
            Cache.";
        reference "ietf-draft-ipfix-mib-10, Section 8
            (ipfixMeteringProcessCacheActiveFlows).";
    }
    leaf unusedCacheEntries {
        type yang:gauge32;
        units flows;
        config false;
        description "The number of unused Cache entries in this
            Cache.";
        reference "ietf-draft-ipfix-mib-10, Section 8
            (ipfixMeteringProcessCacheUnusedCacheEntries).";
    }
}

ct:complex-type NonPermanentCache {
    ct:abstract true;
    ct:extends NonImmediateCache;
```

```
    leaf activeTimeout {
        type uint32;
        units milliseconds;
        description "This parameter configures the time in
            milliseconds after which ... ";
    }
    leaf inactiveTimeout {
        type uint32;
        units milliseconds;
        description "This parameter configures the time in
            milliseconds after which ... ";
    }
}

ct:complex-type NaturalCache {
    if-feature cacheModeNatural;
    ct:extends NonPermanentCache;
}

ct:complex-type TimeoutCache {
    if-feature cacheModeTimeout;
    ct:extends NonPermanentCache;
}

ct:complex-type PermanentCache {
    if-feature cacheModePermanent;
    ct:extends NonImmediateCache;
    leaf exportInterval {
        type uint32;
        units milliseconds;
        description "This parameter configures the interval for
            periodical export of Flow Records in milliseconds.
            If not configured by the user, the Monitoring Device sets
            this parameter.";
    }
}

ct:complex-type ExportDestination {
    ct:abstract true;
    description "Abstract export destination.";
    key name;
    leaf name {
        type nameType;
        description "Key of an export destination.";
    }
}

ct:complex-type IpDestination {
```

```
ct:abstract true;
ct:extends ExportDestination;
description "IP export destination.";
leaf ipfixVersion {
    type uint16;
    default 10;
    description "IPFIX version number.";
}
leaf destinationPort {
    type inet:port-number;
    description "If not configured by the user, the Monitoring
        Device uses the default port number for IPFIX, which is
        4739 without transport layer security and 4740 if transport
        layer security is activated.";
}
choice indexOrName {
    description "Index or name of the interface ... ";
    reference "RFC 1229.";
    leaf ifIndex {
        type uint32;
        description "Index of an interface as stored in the ifTable
            of IF-MIB.";
        reference "RFC 1229.";
    }
    leaf ifName {
        type string;
        description "Name of an interface as stored in the ifTable
            of IF-MIB.";
        reference "RFC 1229.";
    }
}
leaf sendBufferSize {
    type uint32;
    units bytes;
    description "Size of the socket send buffer.
        If not configured by the user, this parameter is set by
        the Monitoring Device.";
}
leaf rateLimit {
    type uint32;
    units "bytes per second";
    description "Maximum number of bytes per second ... ";
    reference "RFC5476, Section 6.3";
}
container transportLayerSecurity {
    presence "If transportLayerSecurity is present, DTLS is
        enabled if the transport protocol is SCTP or UDP, and TLS
        is enabled if the transport protocol is TCP.";
```

```
        description "Transport layer security configuration.";
        uses transportLayerSecurityParameters;
    }
    container transportSession {
        config false;
        description "State parameters of the Transport Session
            directed to the given destination.";
        uses transportSessionParameters;
    }
}

ct:complex-type SctpExporter {
    ct:extends IpDestination;
    description "SCTP exporter.";
    leaf-list sourceIPAddress {
        type inet:ip-address;
        description "List of source IP addresses used ... ";
        reference "RFC 4960 (multi-homed SCTP endpoint).";
    }
    leaf-list destinationIPAddress {
        type inet:ip-address;
        min-elements 1;
        description "One or multiple IP addresses ... ";
        reference "RFC 4960 (multi-homed SCTP endpoint).";
    }
    leaf timedReliability {
        type uint32;
        units milliseconds;
        default 0;
        description "Lifetime in milliseconds ... ";
        reference "RFC 3758; RFC 4960.";
    }
}

ct:complex-type UdpExporter {
    ct:extends IpDestination;
    if-feature udpTransport;
    description "UDP parameters.";
    leaf sourceIPAddress {
        type inet:ip-address;
        description "Source IP address used by the Exporting
            Process ...";
    }
    leaf destinationIPAddress {
        type inet:ip-address;
        mandatory true;
        description "IP address of the Collection Process to which
            IPFIX Messages are sent.";
```

```
}
leaf maxPacketSize {
  type uint16;
  units octets;
  description "This parameter specifies the maximum size of
    IP packets ... ";
}
leaf templateRefreshTimeout {
  type uint32;
  units seconds;
  default 600;
  description "Sets time after which Templates are resent in the
    UDP Transport Session. ... ";
  reference "RFC5101, Section 10.3.6; RFC5815, Section 8
    (ipfixTransportSessionTemplateRefreshTimeout).";
}
leaf optionsTemplateRefreshTimeout {
  type uint32;
  units seconds;
  default 600;
  description "Sets time after which Options Templates are
    resent in the UDP Transport Session. ... ";
  reference "RFC5101, Section 10.3.6; RFC5815, Section 8
    (ipfixTransportSessionOptionsTemplateRefreshTimeout).";
}
leaf templateRefreshPacket {
  type uint32;
  units "IPFIX Messages";
  description "Sets number of IPFIX Messages after which
    Templates are resent in the UDP Transport Session. ... ";
  reference "RFC5101, Section 10.3.6; RFC5815, Section 8
    (ipfixTransportSessionTemplateRefreshPacket).";
}
leaf optionsTemplateRefreshPacket {
  type uint32;
  units "IPFIX Messages";
  description "Sets number of IPFIX Messages after which
    Options Templates are resent in the UDP Transport Session
    protocol. ... ";
  reference "RFC5101, Section 10.3.6; RFC5815, Section 8
    (ipfixTransportSessionOptionsTemplateRefreshPacket).";
}
}

ct:complex-type TcpExporter {
  ct:extends IpDestination;
  if-feature tcpTransport;
  description "TCP exporter";
```

```
    leaf sourceIPAddress {
      type inet:ip-address;
      description "Source IP address used by the Exporting
        Process...";
    }
    leaf destinationIPAddress {
      type inet:ip-address;
      mandatory true;
      description "IP address of the Collection Process to which
        IPFIX Messages are sent.";
    }
  }
}

ct:complex-type FileWriter {
  ct:extends ExportDestination;
  if-feature fileWriter;
  description "File Writer.";
  leaf ipfixVersion {
    type uint16;
    default 10;
    description "IPFIX version number.";
  }
  leaf file {
    type inet:uri;
    mandatory true;
    description "URI specifying the location of the file.";
  }
  leaf bytes {
    type yang:counter64;
    units octets;
    config false;
    description "The number of bytes written by the File
      Writer...";
  }
  leaf messages {
    type yang:counter64;
    units "IPFIX Messages";
    config false;
    description "The number of IPFIX Messages written by the File
      Writer. ... ";
  }
  leaf discardedMessages {
    type yang:counter64;
    units "IPFIX Messages";
    config false;
    description "The number of IPFIX Messages that could not be
      written by the File Writer ... ";
  }
}
```

```
leaf records {
  type yang:counter64;
  units "Data Records";
  config false;
  description "The number of Data Records written by the File
    Writer. ... ";
}
leaf templates {
  type yang:counter32;
  units "Templates";
  config false;
  description "The number of Template Records (excluding
    Options Template Records) written by the File Writer.
    ... ";
}
leaf optionsTemplates {
  type yang:counter32;
  units "Options Templates";
  config false;
  description "The number of Options Template Records written
    by the File Writer. ... ";
}
leaf fileWriterDiscontinuityTime {
  type yang:date-and-time;
  config false;
  description "Timestamp of the most recent occasion at which
    one or more File Writer counters suffered a discontinuity.
    ... ";
}
list template {
  config false;
  description "This list contains the Templates and Options
    Templates that have been written by the File Reader. ... ";
  uses templateParameters;
}
}

ct:complex-type ExportingProcess {
  if-feature exporter;
  description "Exporting Process of the Monitoring Device.";
  key name;
  leaf name {
    type nameType;
    description "Key of this list.";
  }
  leaf exportMode {
    type identityref {
      base "exportMode";
    }
  }
}
```

```
    }
    default parallel;
    description "This parameter determines to which configured
        destination(s) the incoming Data Records are exported.";
}
ct:instance-list destination {
    ct:instance-type ExportDestination;
    min-elements 1;
    description "Export destinations.";
}
list options {
    key name;
    description "List of options reported by the Exporting
        Process.";
    leaf name {
        type nameType;
        description "Key of this list.";
    }
    leaf optionsType {
        type identityref {
            base "optionsType";
        }
        mandatory true;
        description "Type of the exported options data.";
    }
    leaf optionsTimeout {
        type uint32;
        units milliseconds;
        description "Time interval for periodic export of the options
            data. ... ";
    }
}
}

ct:complex-type CollectingProcess {
    description "A Collecting Process.";
    key name;
    leaf name {
        type nameType;
        description "Key of a collecting process.";
    }
    ct:instance-list sctpCollector {
        ct:instance-type SctpCollector;
        description "List of SCTP receivers (sockets) on which the
            Collecting Process receives IPFIX Messages.";
    }
    ct:instance-list udpCollector {
        if-feature udpTransport;
    }
}
```



```
    ct:instance-type UdpCollector;
    description "List of UDP receivers (sockets) on which the
        Collecting Process receives IPFIX Messages.";
}
ct:instance-list tcpCollector {
    if-feature tcpTransport;
    ct:instance-type TcpCollector;
    description "List of TCP receivers (sockets) on which the
        Collecting Process receives IPFIX Messages.";
}
ct:instance-list fileReader {
    if-feature fileReader;
    ct:instance-type FileReader;
    description "List of File Readers from which the Collecting
        Process reads IPFIX Messages.";
}
leaf-list exportingProcess {
    type instance-identifier { ct:instance-type ExportingProcess; }
    description "Export of received records without any
        modifications. Records are processed by all Exporting
        Processes in the list.";
}
}

ct:complex-type Collector {
    ct:abstract true;
    description "Abstract collector.";
    key name;
    leaf name {
        type nameType;
        description "Key of collectors";
    }
}

ct:complex-type IpCollector {
    ct:abstract true;
    ct:extends Collector;
    description "Collector for IP transport protocols.";
    leaf localPort {
        type inet:port-number;
        description "If not configured, the Monitoring Device uses the
            default port number for IPFIX, which is 4739 without
            transport layer security and 4740 if transport layer
            security is activated.";
    }
    container transportLayerSecurity {
        presence "If transportLayerSecurity is present, DTLS is enabled
            if the transport protocol is SCTP or UDP, and TLS is enabled
```

```
        if the transport protocol is TCP.";
        description "Transport layer security configuration.";
        uses transportLayerSecurityParameters;
    }
    list transportSession {
        config false;
        description "This list contains the currently established
            Transport Sessions terminating at the given socket.";
        uses transportSessionParameters;
    }
}

ct:complex-type SctpCollector {
    ct:extends IpCollector;
    description "Collector listening on aSCTP socket";
    leaf-list localIPAddress {
        type inet:ip-address;
        description "List of local IP addresses ... ";
        reference "RFC 4960 (multi-homed SCTP endpoint).";
    }
}

ct:complex-type UdpCollector {
    ct:extends IpCollector;
    description "Parameters of a listening UDP socket at a
        Collecting Process.";
    leaf-list localIPAddress {
        type inet:ip-address;
        description "List of local IP addresses on which the Collecting
            Process listens for IPFIX Messages.";
    }
    leaf templateLifeTime {
        type uint32;
        units seconds;
        default 1800;
        description "Sets the lifetime of Templates for all UDP
            Transport Sessions ... ";
        reference "RFC5101, Section 10.3.7; RFC5815, Section 8
            (ipfixTransportSessionTemplateRefreshTimeout).";
    }
    leaf optionsTemplateLifeTime {
        type uint32;
        units seconds;
        default 1800;
        description "Sets the lifetime of Options Templates for all
            UDP Transport Sessions terminating at this UDP socket.
            ... ";
        reference "RFC5101, Section 10.3.7; RFC5815, Section 8
```

```
        (ipfixTransportSessionOptionsTemplateRefreshTimeout).";
    }
    leaf templateLifePacket {
        type uint32;
        units "IPFIX Messages";
        description "If this parameter is configured, Templates
            defined in a UDP Transport Session become invalid if ...";
        reference "RFC5101, Section 10.3.7; RFC5815, Section 8
            (ipfixTransportSessionTemplateRefreshPacket).";
    }
    leaf optionsTemplateLifePacket {
        type uint32;
        units "IPFIX Messages";
        description "If this parameter is configured, Options
            Templates defined in a UDP Transport Session become
            invalid if ...";
        reference "RFC5101, Section 10.3.7; RFC5815, Section 8
            (ipfixTransportSessionOptionsTemplateRefreshPacket).";
    }
}

ct:complex-type TcpCollector {
    ct:extends IpCollector;
    description "Collector listening on a TCP socket.";
    leaf-list localIpAddress {
        type inet:ip-address;
        description "List of local IP addresses on which the Collecting
            Process listens for IPFIX Messages.";
    }
}

ct:complex-type FileReader {
    ct:extends Collector;
    description "File Reading collector.";
    leaf file {
        type inet:uri;
        mandatory true;
        description "URI specifying the location of the file.";
    }
    leaf bytes {
        type yang:counter64;
        units octets;
        config false;
        description "The number of bytes read by the File Reader.
            ... ";
    }
    leaf messages {
        type yang:counter64;
    }
}
```

```
    units "IPFIX Messages";
    config false;
    description "The number of IPFIX Messages read by the File
        Reader. ... ";
}
leaf records {
    type yang:counter64;
    units "Data Records";
    config false;
    description "The number of Data Records read by the File
        Reader. ... ";
}
leaf templates {
    type yang:counter32;
    units "Templates";
    config false;
    description "The number of Template Records (excluding
        Options Template Records) read by the File Reader. ...";
}
leaf optionsTemplates {
    type yang:counter32;
    units "Options Templates";
    config false;
    description "The number of Options Template Records read by
        the File Reader. ... ";
}
leaf fileReaderDiscontinuityTime {
    type yang:date-and-time;
    config false;
    description "Timestamp of the most recent occasion ... ";
}
list template {
    config false;
    description "This list contains the Templates and Options
        Templates that have been read by the File Reader.
        Withdrawn or invalidated (Options) Template MUST be removed
        from this list.";
    uses templateParameters;
}
}

ct:complex-type SelectionProcess {
    description "Selection Process";
    key name;
    leaf name {
        type nameType;
        description "Key of a selection process.";
    }
}
```

```

ct:instance-list selector {
  ct:instance-type Selector;
  min-elements 1;
  ordered-by user;
  description "List of Selectors that define the action of the
    Selection Process on a single packet. The Selectors are
    serially invoked in the same order as they appear in this
    list.";
}
list selectionSequence {
  config false;
  description "This list contains the Selection Sequence IDs
    which are assigned by the Monitoring Device ... ";
  reference "RFC5476.";
  leaf observationDomainId {
    type uint32;
    description "Observation Domain ID for which the
      Selection Sequence ID is assigned.";
  }
  leaf selectionSequenceId {
    type uint64;
    description "Selection Sequence ID used in the Selection
      Sequence (Statistics) Report Interpretation.";
  }
}
leaf cache {
  type instance-identifier { ct:instance-type Cache; }
  description "Cache which receives the output of the
    Selection Process.";
}
}

/*****
* Groupings
*****/

grouping transportLayerSecurityParameters {
  description "Transport layer security parameters.";
  leaf-list localCertificationAuthorityDN {
    type string;
    description "Distinguished names of certification authorities
      whose certificates may be used to identify the local
      endpoint.";
  }
  leaf-list localSubjectDN {
    type string;
    description "Distinguished names which may be used in the
      certificates to identify the local endpoint.";
  }
}

```

```
}
leaf-list localSubjectFQDN {
  type inet:domain-name;
  description "Fully qualified domain names which may be used to
    in the certificates to identify the local endpoint.";
}
leaf-list remoteCertificationAuthorityDN {
  type string;
  description "Distinguished names of certification authorities
    whose certificates are accepted to authorize remote
    endpoints.";
}
leaf-list remoteSubjectDN {
  type string;
  description "Distinguished names which are accepted in
    certificates to authorize remote endpoints.";
}
leaf-list remoteSubjectFQDN {
  type inet:domain-name;
  description "Fully qualified domain name which are accepted in
    certificates to authorize remote endpoints.";
}
}

grouping templateParameters {
  description "State parameters of a Template used by an Exporting
    Process or received by a Collecting Process ... ";
  reference "RFC5101; RFC5815, Section 8 (ipfixTemplateEntry,
    ipfixTemplateDefinitionEntry, ipfixTemplateStatsEntry)";
  leaf observationDomainId {
    type uint32;
    description "The ID of the Observation Domain for which this
      Template is defined.";
    reference "RFC5815, Section 8
      (ipfixTemplateObservationDomainId).";
  }
  leaf templateId {
    type uint16 {
      range "256..65535" {
        description "Valid range of Template IDs.";
        reference "RFC5101";
      }
    }
    description "This number indicates the Template Id in the IPFIX
      message.";
    reference "RFC5815, Section 8 (ipfixTemplateId).";
  }
  leaf setId {
```

```
    type uint16;
    description "This number indicates the Set ID of the Template.
    ... ";
    reference "RFC5815, Section 8 (ipfixTemplateSetId).";
}
leaf accessTime {
    type yang:date-and-time;
    description "Used for Exporting Processes, ... ";
    reference "RFC5815, Section 8 (ipfixTemplateAccessTime).";
}
leaf templateDataRecords {
    type yang:counter64;
    description "The number of transmitted or received Data
    Records ... ";
    reference "RFC5815, Section 8 (ipfixTemplateDataRecords).";
}
leaf templateDiscontinuityTime {
    type yang:date-and-time;
    description "Timestamp of the most recent occasion at which
    the counter templateDataRecords suffered a discontinuity.
    ... ";
    reference "RFC5815, Section 8
    (ipfixTemplateDiscontinuityTime).";
}
list field {
    description "This list contains the (Options) Template
    fields of which the (Options) Template is defined.
    ... ";
    leaf ieId {
        type uint16 {
            range "1..32767" {
                description "Valid range of Information Element
                identifiers.";
                reference "RFC5102, Section 4.";
            }
        }
        description "This parameter indicates the Information
        Element Id of the field.";
        reference "RFC5815, Section 8 (ipfixTemplateDefinitionIeId);
        RFC5102.";
    }
    leaf ieLength {
        type uint16;
        units octets;
        description "This parameter indicates the length of the
        Information Element of the field.";
        reference "RFC5815, Section 8
        (ipfixTemplateDefinitionIeLength); RFC5102.";
    }
}
```

```
    }
    leaf ieEnterpriseNumber {
        type uint32;
        description "This parameter indicates the IANA enterprise
            number of the authority ... ";
        reference "RFC5815, Section 8
            (ipfixTemplateDefinitionIeEnterpriseNumber).";
    }
    leaf isFlowKey {
        when "../..../setId = 2" {
            description "This parameter is available for non-Options
                Templates (Set ID is 2).";
        }
        type empty;
        description "If present, this is a Flow Key field.";
        reference "RFC5815, Section 8
            (ipfixTemplateDefinitionFlags).";
    }
    leaf isScope {
        when "../..../setId = 3" {
            description "This parameter is available for Options
                Templates (Set ID is 3).";
        }
        type empty;
        description "If present, this is a scope field.";
        reference "RFC5815, Section 8
            (ipfixTemplateDefinitionFlags).";
    }
}

grouping transportSessionParameters {
    description "State parameters of a Transport Session ... ";
    reference "RFC5101, RFC5815, Section 8
        (ipfixTransportSessionEntry,
            ipfixTransportSessionStatsEntry)";
    leaf ipfixVersion {
        type uint16;
        description "Used for Exporting Processes, this parameter
            contains the version number of the IPFIX protocol ... ";
        reference "RFC5815, Section 8
            (ipfixTransportSessionIpfixVersion).";
    }
    leaf sourceAddress {
        type inet:ip-address;
        description "The source address of the Exporter of the
            IPFIX Transport Session... ";
        reference "RFC5815, Section 8
```



```
        (ipfixTransportSessionSourceAddressType,  
        ipfixTransportSessionSourceAddress).";  
    }  
    leaf destinationAddress {  
        type inet:ip-address;  
        description "The destination address of the Collector of  
            the IPFIX Transport Session... ";  
        reference "RFC5815, Section 8  
            (ipfixTransportSessionDestinationAddressType,  
            ipfixTransportSessionDestinationAddress).";  
    }  
    leaf sourcePort {  
        type inet:port-number;  
        description "The transport protocol port number of the  
            Exporter of the IPFIX Transport Session.";  
        reference "RFC5815, Section 8  
            (ipfixTransportSessionSourcePort).";  
    }  
    leaf destinationPort {  
        type inet:port-number;  
        description "The transport protocol port number of the  
            Collector of the IPFIX Transport Session... ";  
        reference "RFC5815, Section 8  
            (ipfixTransportSessionDestinationPort).";  
    }  
    leaf sctpAssocId {  
        type uint32;  
        description "The association id used for the SCTP session  
            between the Exporter and the Collector ... ";  
        reference "RFC5815, Section 8  
            (ipfixTransportSessionSctpAssocId),  
            RFC3871";  
    }  
    leaf status {  
        type transportSessionStatus;  
        description "Status of the Transport Session.";  
        reference "RFC5815, Section 8 (ipfixTransportSessionStatus).";  
    }  
    leaf rate {  
        type yang:gauge32;  
        units "bytes per second";  
        description "The number of bytes per second transmitted by the  
            Exporting Process or received by the Collecting Process.  
            This parameter is updated every second.";  
        reference "RFC5815, Section 8 (ipfixTransportSessionRate).";  
    }  
    leaf bytes {  
        type yang:counter64;
```

```
    units bytes;
    description "The number of bytes transmitted by the
        Exporting Process or received by the Collecting
        Process ... ";
    reference "RFC5815, Section 8 (ipfixTransportSessionBytes).";
}
leaf messages {
    type yang:counter64;
    units "IPFIX Messages";
    description "The number of messages transmitted by the
        Exporting Process or received by the Collecting Process... ";
    reference "RFC5815, Section 8
        (ipfixTransportSessionMessages).";
}
leaf discardedMessages {
    type yang:counter64;
    units "IPFIX Messages";
    description "Used for Exporting Processes, this parameter
        indicates the number of messages that could not be
        sent ...";
    reference "RFC5815, Section 8
        (ipfixTransportSessionDiscardedMessages).";
}
leaf records {
    type yang:counter64;
    units "Data Records";
    description "The number of Data Records transmitted ... ";
    reference "RFC5815, Section 8
        (ipfixTransportSessionRecords).";
}
leaf templates {
    type yang:counter32;
    units "Templates";
    description "The number of Templates transmitted by the
        Exporting Process or received by the Collecting Process.
        ... ";
    reference "RFC5815, Section 8
        (ipfixTransportSessionTemplates).";
}
leaf optionsTemplates {
    type yang:counter32;
    units "Options Templates";
    description "The number of Option Templates transmitted by the
        Exporting Process or received by the Collecting Process...";
    reference "RFC5815, Section 8
        (ipfixTransportSessionOptionsTemplates).";
}
leaf transportSessionStartTime {
```

```
    type yang:date-and-time;
    description "Timestamp of the start of the given Transport
        Session... ";
}
leaf transportSessionDiscontinuityTime {
    type yang:date-and-time;
    description "Timestamp of the most recent occasion at which
        one or more of the Transport Session counters suffered a
        discontinuity... ";
    reference "RFC5815, Section 8
        (ipfixTransportSessionDiscontinuityTime).";
}
list template {
    description "This list contains the Templates and Options
        Templates that are transmitted by the Exporting Process
        or received by the Collecting Process.
        Withdrawn or invalidated (Options) Template MUST be removed
        from this list.";
    uses templateParameters;
}
}

/*****
* Main container
*****/

container ipfix {
    description "Top-level node of the IPFIX/PSAMP configuration
        data model.";

    ct:instance-list collectingProcess {
        if-feature collector;
        ct:instance-type CollectingProcess;
    }

    ct:instance-list observationPoint {
        if-feature meter;
        ct:instance-type ObservationPoint;
    }

    ct:instance-list selectionProcess {
        if-feature meter;
        ct:instance-type SelectionProcess;
    }

    ct:instance-list cache {
        if-feature meter;
        description "Cache of the Monitoring Device.";
    }
}
```

```
        ct:instance-type Cache;
    }

    ct:instance-list exportingProcess {
        if-feature exporter;
        description "Exporting Process of the Monitoring Device.";
        ct:instance-type ExportingProcess;
    }
}
}
<CODE ENDS>
```

Authors' Addresses

Bernd Linowski
TCS/Nokia Siemens Networks
Heltorfer Strasse 1
Duesseldorf 40472
Germany

EMail: bernd.linowski@ext.nsn.com

Mehmet Ersue
Nokia Siemens Networks
St.-Martin-Strasse 53
Munich 81541
Germany

EMail: mehmet.ersue@nsn.com

Siarhei Kuryla
360 Treasury Systems
Grueneburgweg 16-18
Frankfurt am Main 60322
Germany

EMail: s.kuryla@gmail.com

