
A Guide to NETCONF for SNMP Developers

Andy Bierman <andy@yumaworks.com>

v0.6 2014-07-10

Abstract

- NETCONF is a standards track protocol developed in the IETF, and YANG is the associated data modeling language
- Recently the IESG recommended the usage of NETCONF and YANG for new management work in the IETF that involves configuration management operations
 - <http://www.ietf.org/iesg/statement/writable-mib-module.html>
- This tutorial covers the NETCONF and YANG concepts, and highlights the differences and advantages of YANG/NETCONF over SMIv2/SNMP
 - Some knowledge of SNMP and SMIv2 is required

Contents

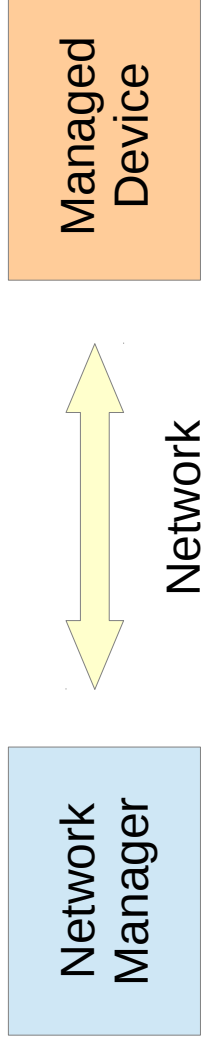
- Overview
 - Comparison of SNMP/SMIPv2 and NETCONF/YANG properties
- NETCONF Basics
 - Overview of core NETCONF protocol concepts
- YANG Basics
 - Overview of core YANG data modeling concepts
- Advanced Topics
 - Overview of advanced NETCONF and YANG concepts

Part 1

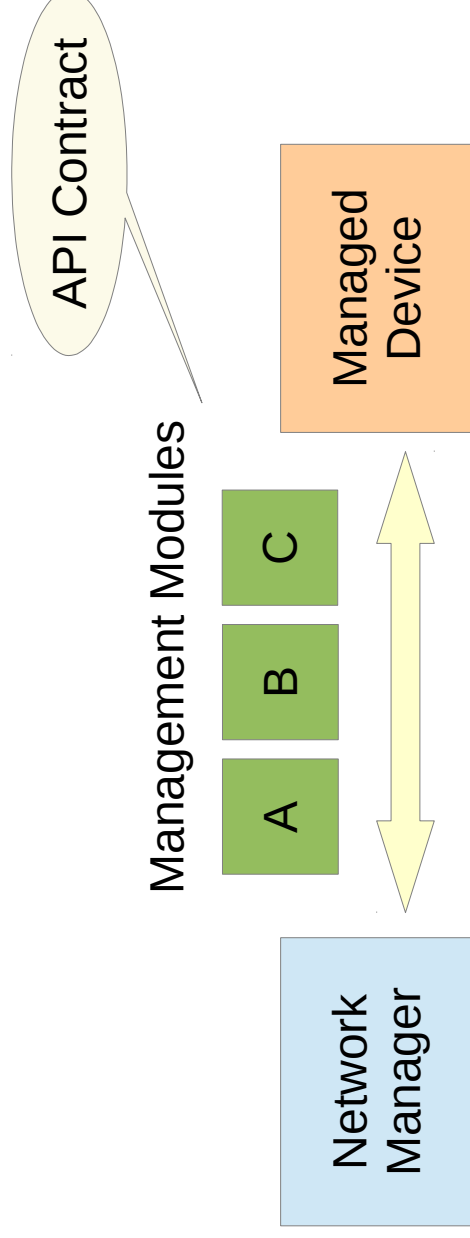
- Overview
 - Comparison of SNMP/SMIPv2 and NETCONF/YANG properties
- NETCONF Basics
 - Overview of core NETCONF protocol concepts
- YANG Basics
 - Overview of core YANG data modeling concepts
- Advanced Topics
 - Overview of advanced NETCONF and YANG concepts

Management Framework

NETCONF and SNMP are basically the same at 10,000 feet

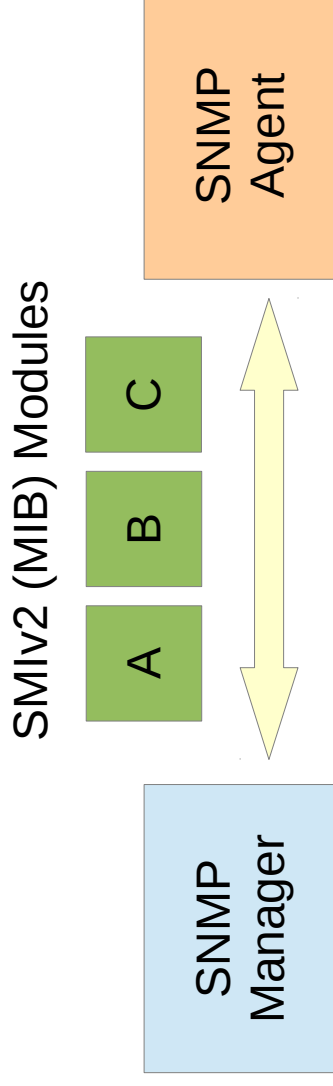


The schema for shared management information is defined independently of the protocol, using a modular data modeling language

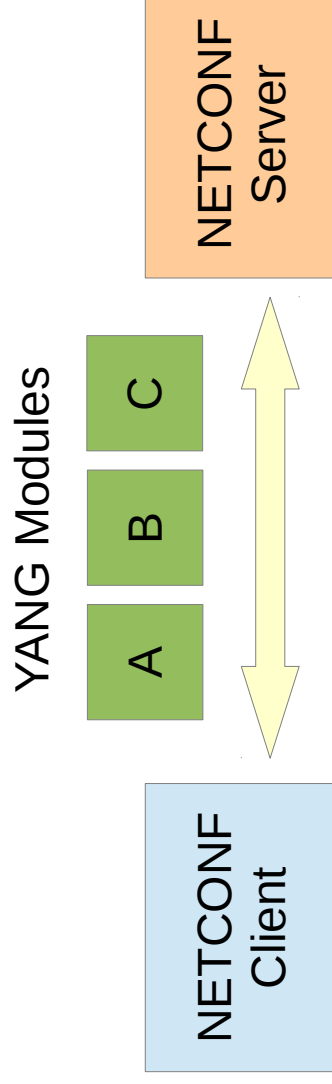


Management Framework (2)

SNMP

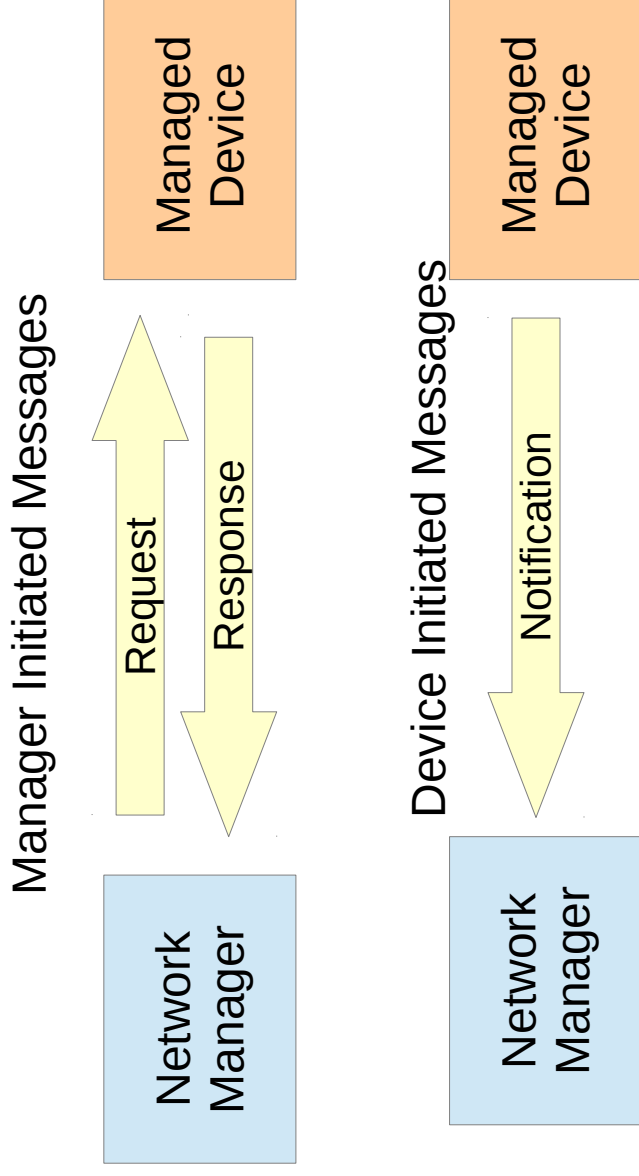


NETCONF



Management Protocol

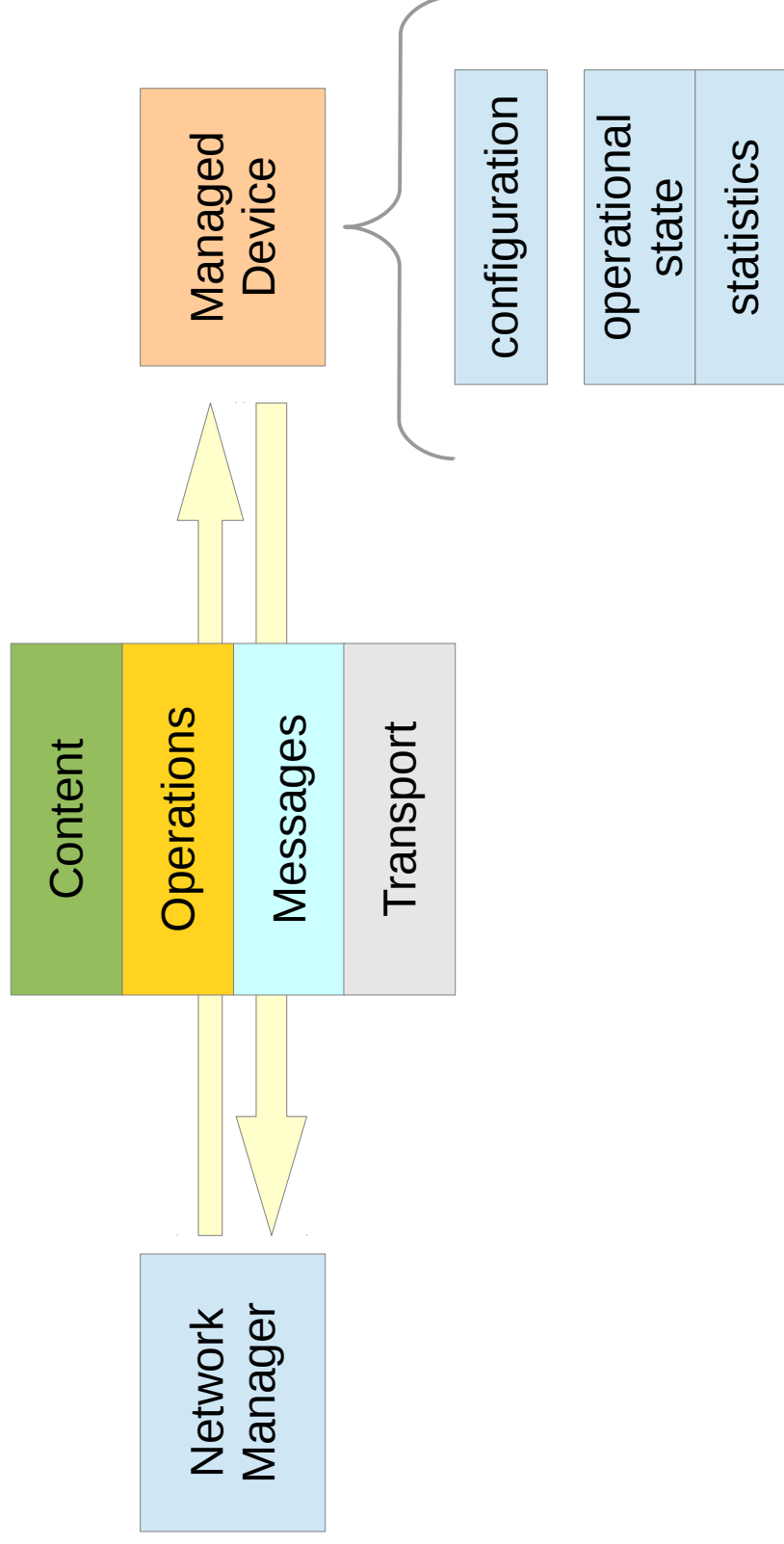
NETCONF and SNMP have similar protocol message models



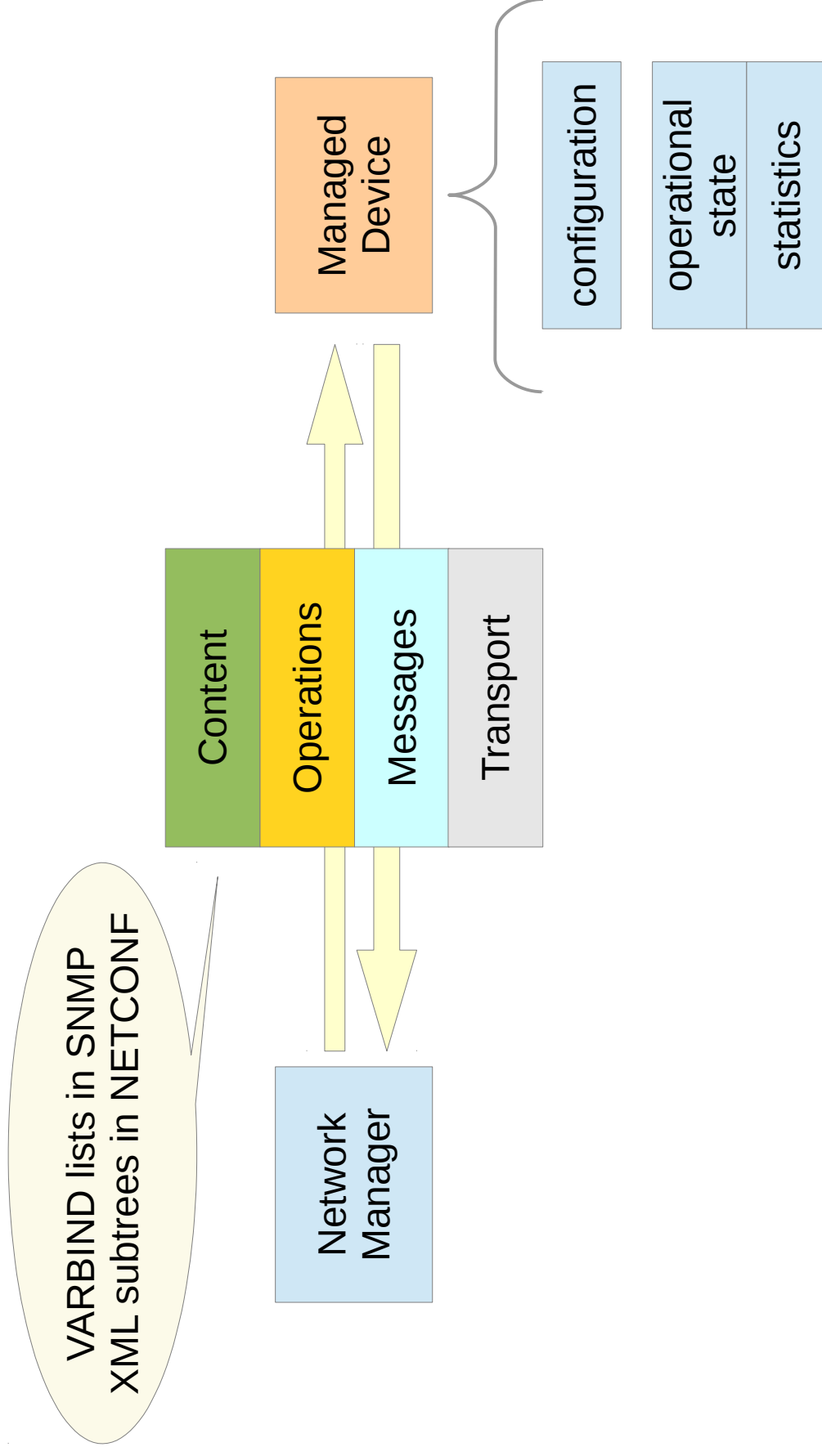
Why Bother to Upgrade?

- Configuration Management
 - Robust transaction model better for clients
 - Validation and rollback features
 - Network-wide commits possible
- Next Generation Features
 - Advanced data modeling with YANG
 - Custom protocol operations defined with YANG
 - Custom language statements to support data-driven tools
 - Advanced retrieval filtering (XML subtree and XPath)

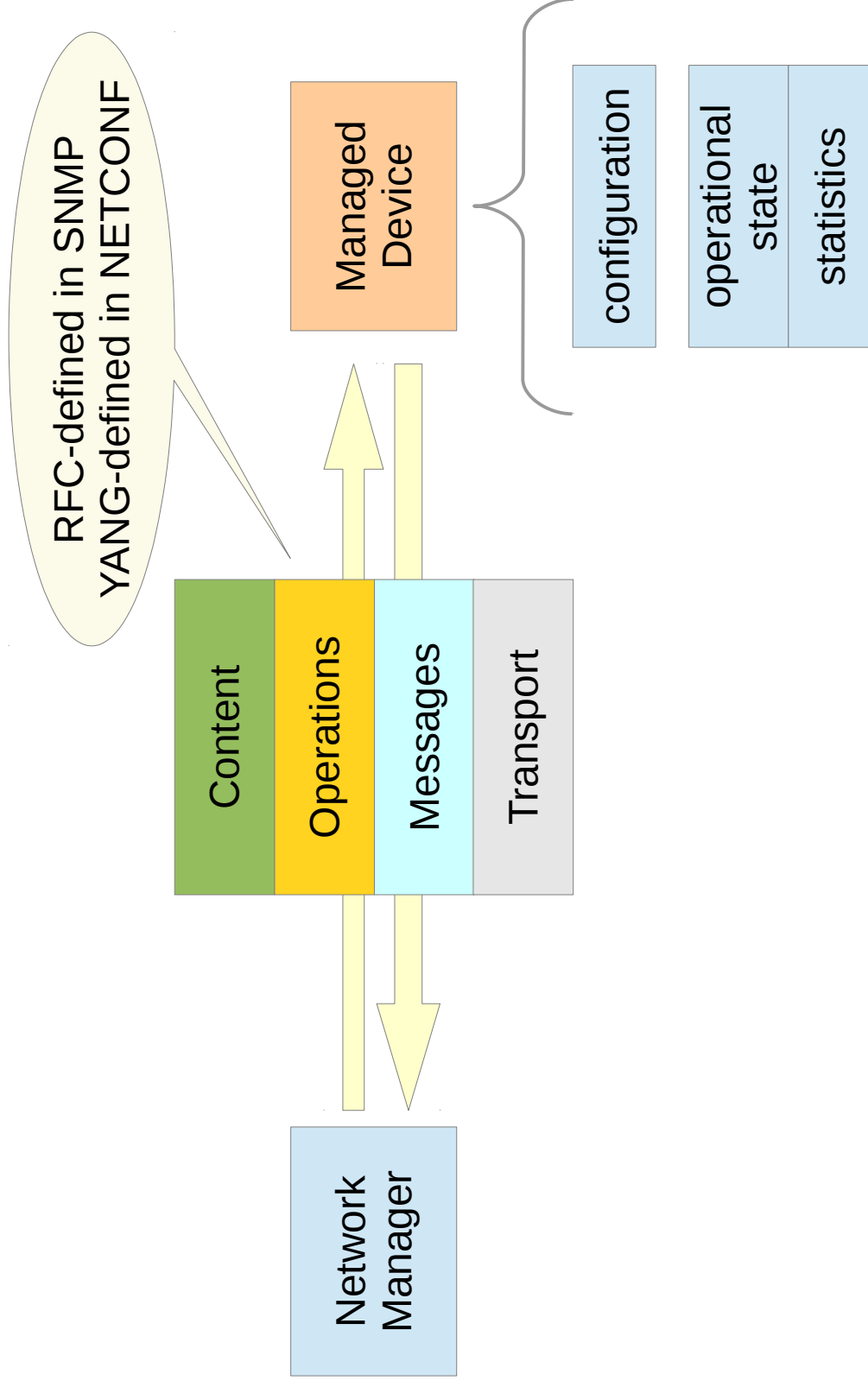
Protocol Layers



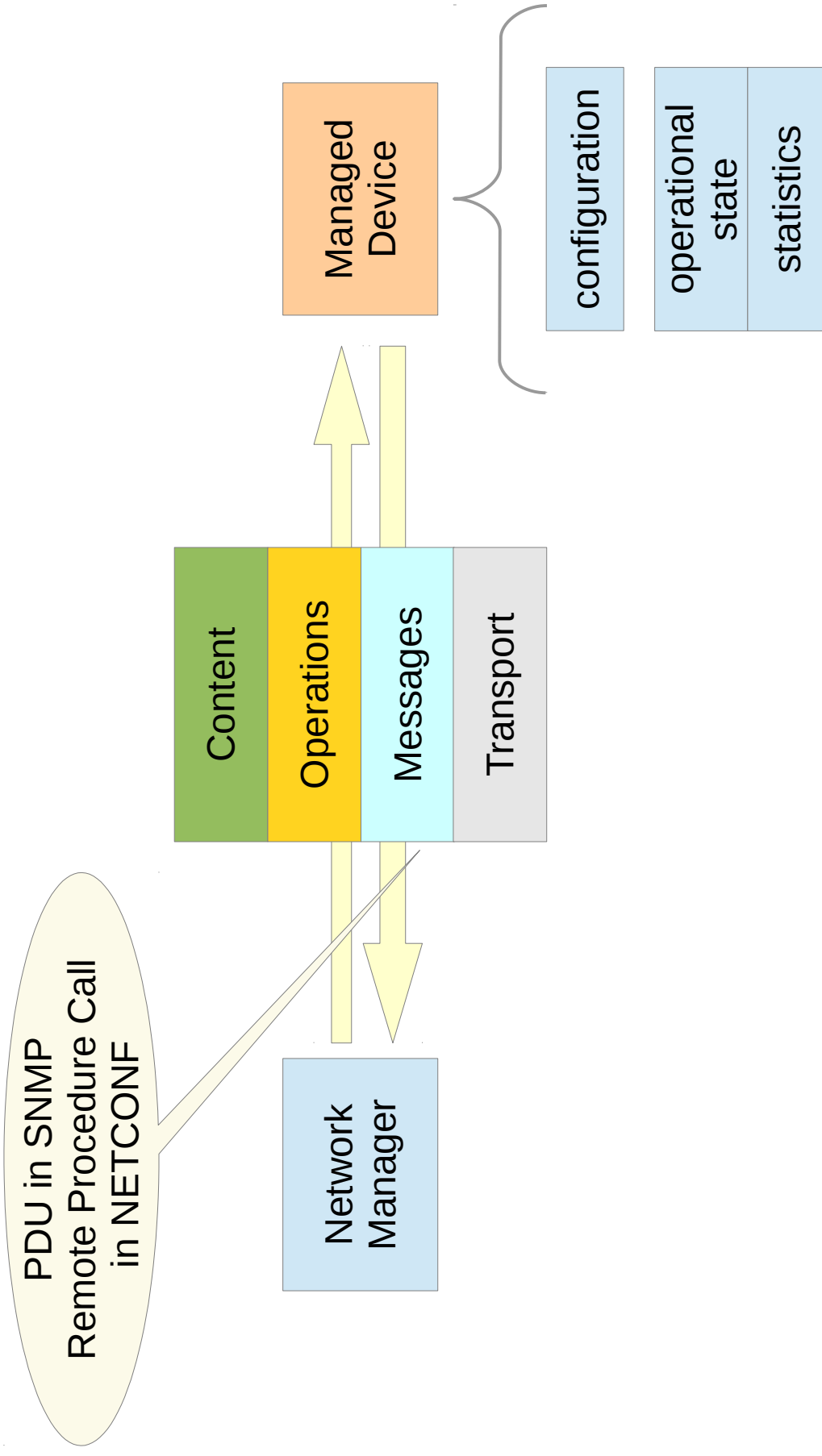
Protocol Layers: Content



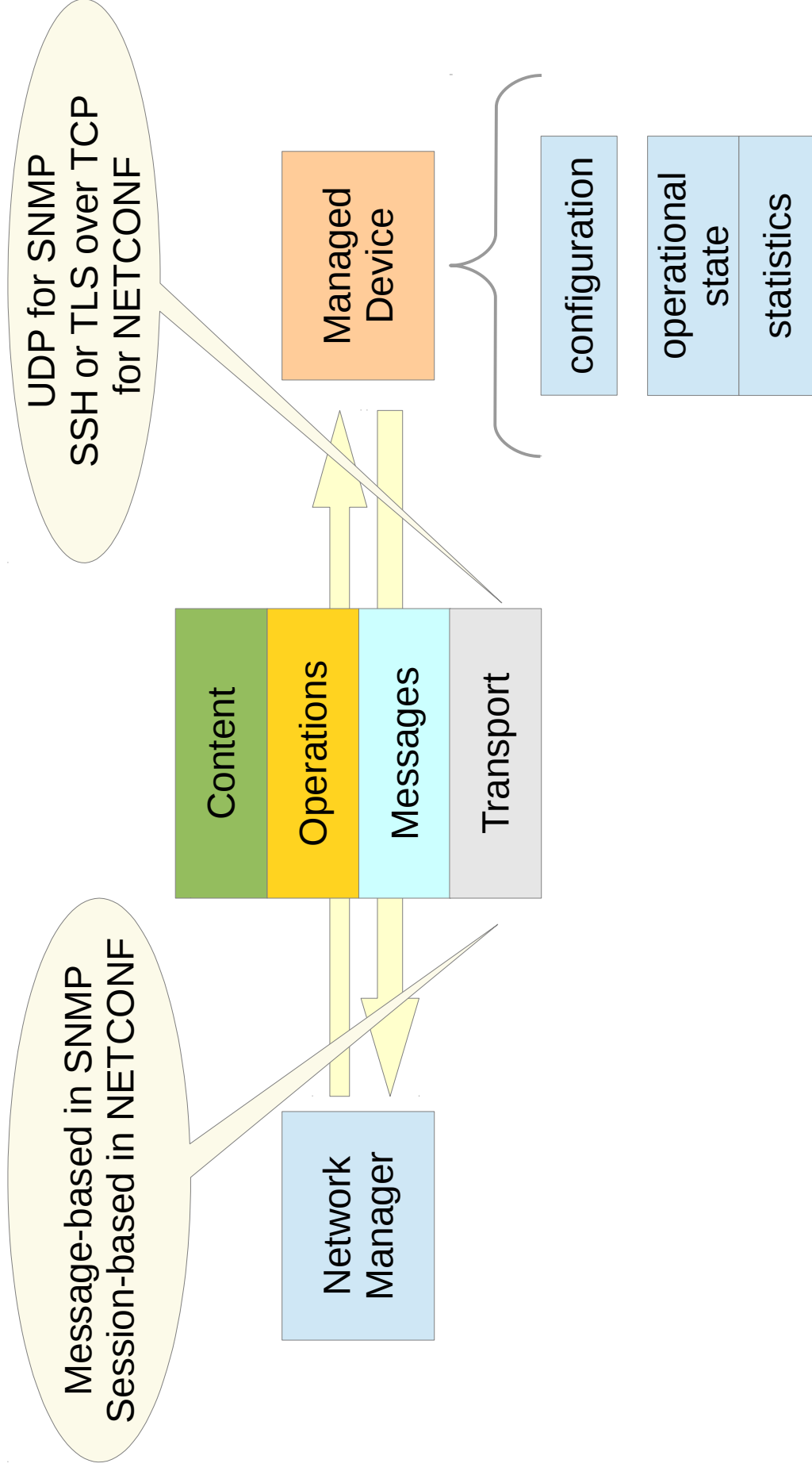
Protocol Layers: Operations



Protocol Layers: Messages



Protocol Layers: Transport

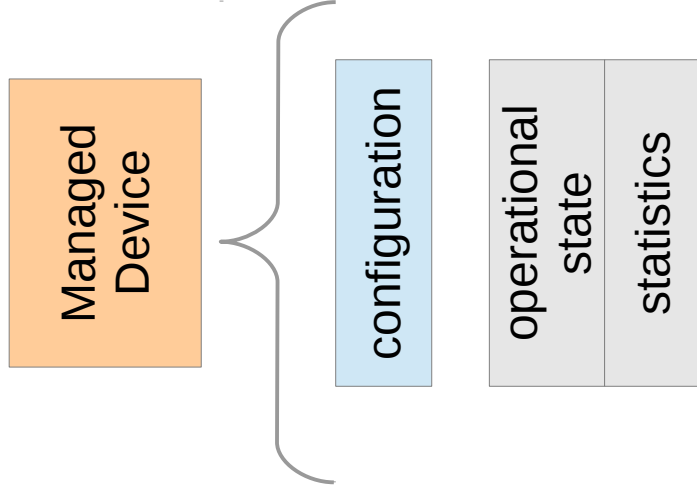


Protocol Layer Comparison

Layer	SNMP/SMIv2	NETCONF/YANG
Content Device Data Notification Data	OBJECT-TYPE NOTIFICATION-TYPE	data-def-stmt notification-stmt
Operations	Set, Get, GetNext, GetBulk	rpc-stmt (edit-config, copy-config)
Messages Device Data Notification Data	Set, Get, GetNext, GetBulk PDUs Trap PDU Inform PDU Report PDU	<rpc>, <rpc-reply> <notification> N/A N/A
Transport	Message-based: UDP	Session-based: SSH/TCP

Management Data

- NETCONF configuration

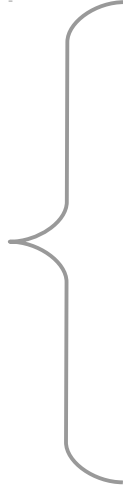


- **datastore:** A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.
- **configuration datastore:** The datastore holding the complete set of configuration data that is required to get a device from its initial default state into a desired operational state.
- **configuration data:** The set of writable data that is required to transform a system from its initial default state into its current state.

Management Data (2)

- NETCONF state data

Managed
Device

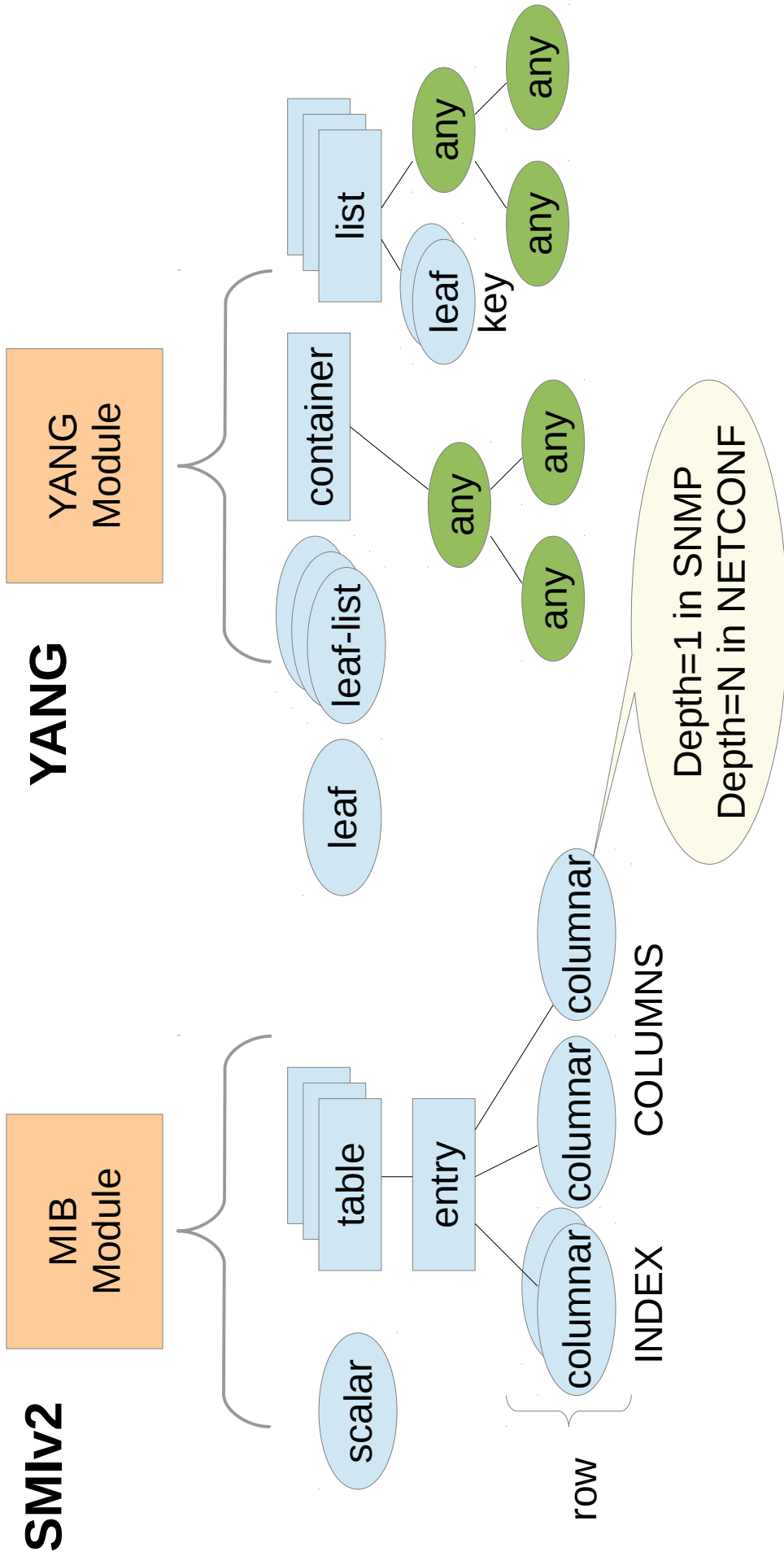


configuration

operational state
statistics

- **state data:** The additional data on a system that is not configuration data such as read-only status information and collected statistics
- Separation of configuration and state/statistics motivated by operator requirements gathered at IAB Workshop on Network Management
 - RFC 3535, section 3
- YANG XPath validation explicitly separates the configuration and non-configuration data nodes

Basic Data Building Blocks





Data Naming

SMIv2

MIB
Module

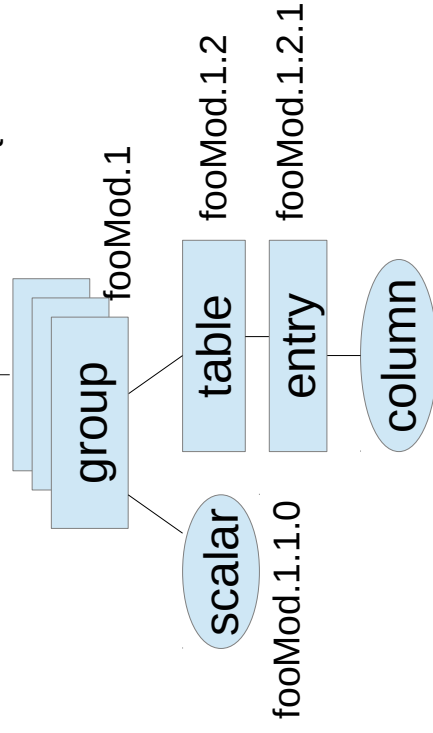
YANG

YANG
Module

OBJECT IDENTIFIER Naming

MODULE-IDENTITY
(OID)

fooMod ::= { mib-2 x }

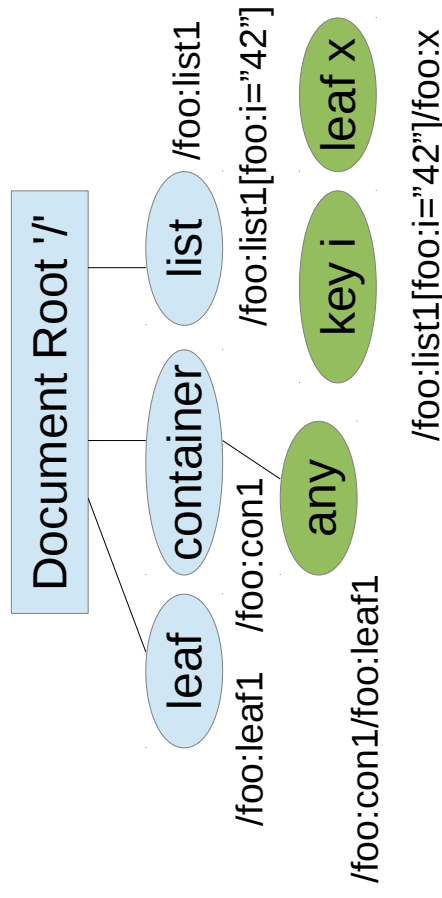


fooMod.1.2.1.1.<INDEX>

XPath Absolute Path Naming

namespace-stmt
(URI)

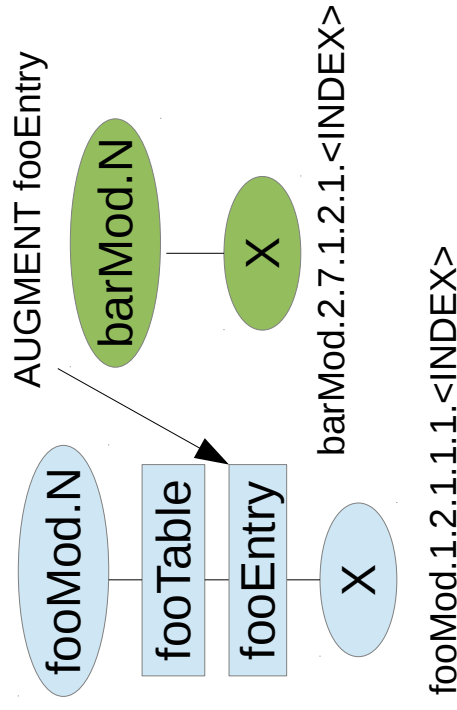
namespace urn:ietf:params:xml:ns:yang:foo



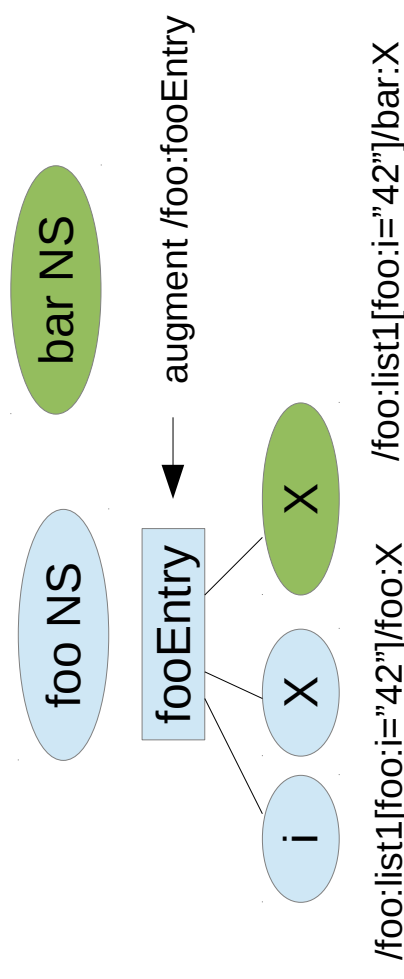
Data Organization

- Distributed naming authority in both SNMP and NETCONF (Naming collisions are not possible between SDOs and/or vendors)
 - Augment handled differently
 - SMIv2 distributes this data, each AUGMENT in own OID subtree
 - YANG maintains all augmenting data within the augmented subtree

SMIv2

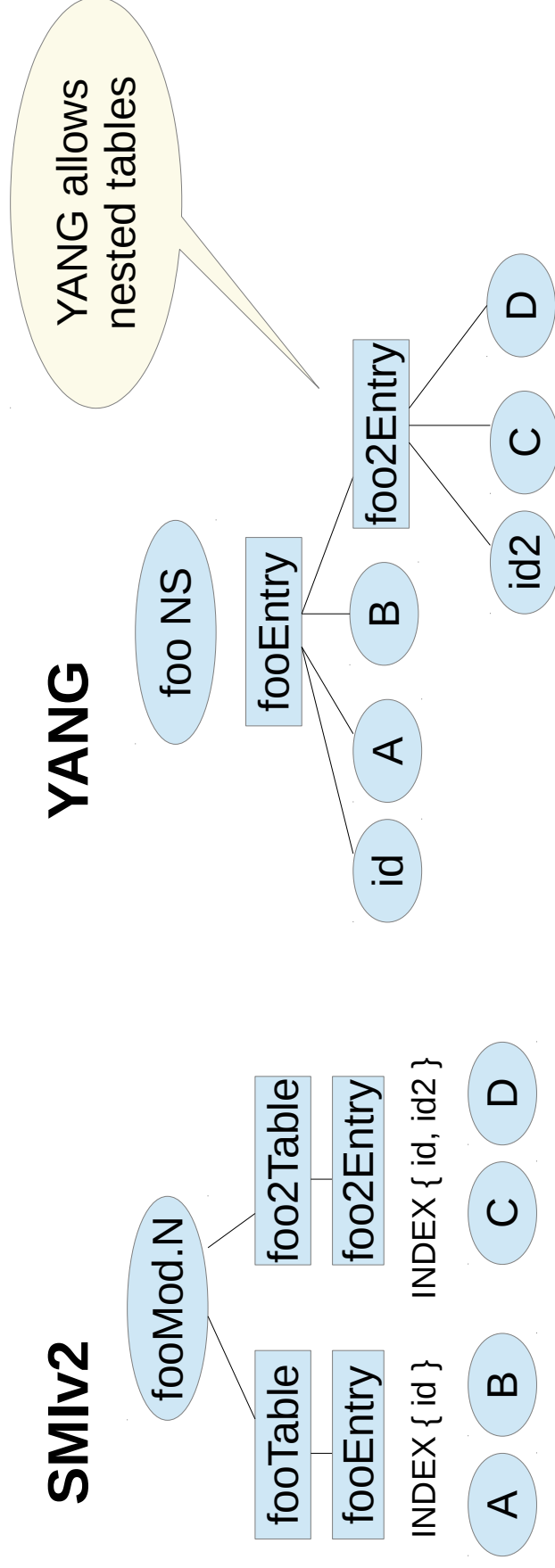


YANG



Data Translation

- Translation between SMIv2 and YANG defined in RFC 6643
- Related SMIv2 tables can be translated as flat lists or combined as nested lists in YANG
- Provides a standard read-only translation of SMIv2 data that can be retrieved with NETCONF

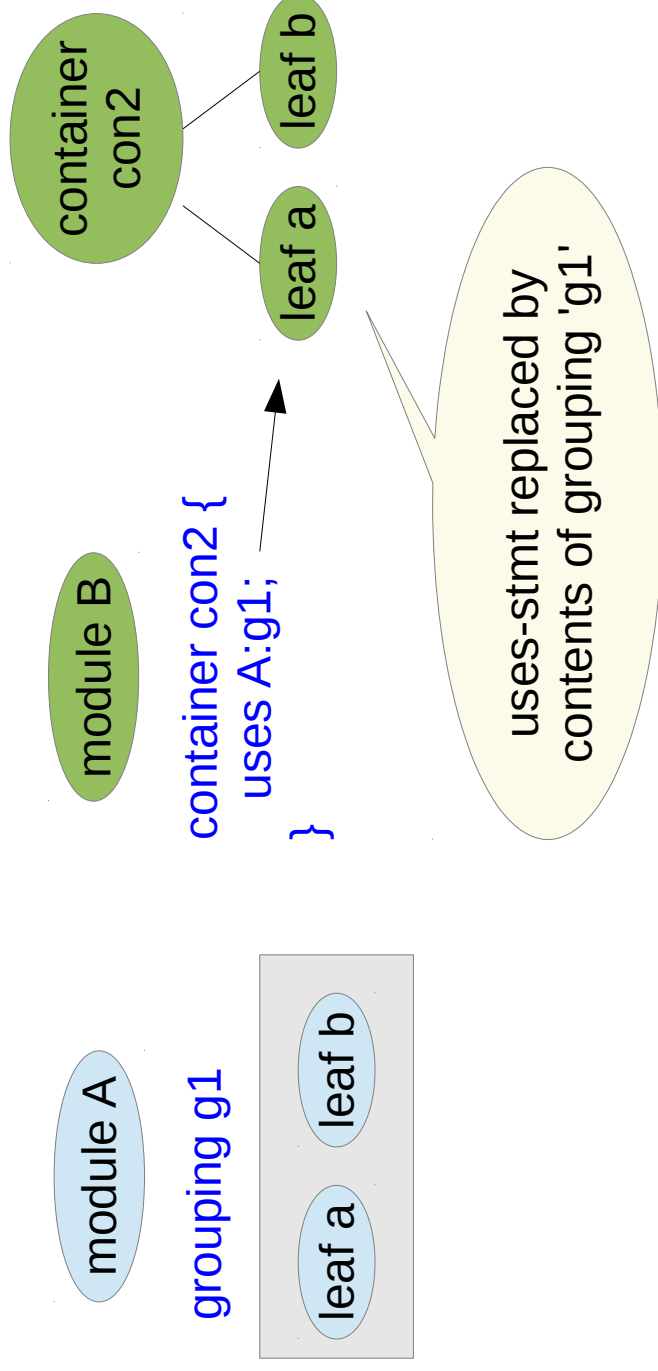


SMIPv2 to YANG Mapping Issues

- What if there are writable objects in the MIB module?
 - Could hand-edit the RFC 6643 output and add configuration data nodes
 - Could hand-edit an additional module that uses the RFC 6643 output
- What if the MIB data naming does not align with new YANG data structures?
 - Could hand-edit the RFC 6643 output and change the data naming
 - Could cut-and-paste the RFC 6643 output for leaf definitions into the new YANG module
- Often new data structures are created instead of using a mix of hand-edited and auto-generated YANG modules
 - YANG Interface Management (RFC 7723) took this approach
 - Counter discontinuity-time changed from TimeStamp to date-and-time
 - No TimeTicks in NETCONF; XSD 'dateTime' timestamp format used instead

Data Reuse

- NETCONF allows data to be defined in reusable groupings
- **grouping-stmt** defines abstract group of objects
 - no namespace assigned to objects in the grouping
- **uses-stmt** defines a concrete expansion point for a grouping
 - given the namespace of the module containing the uses-stmt



Getting Started

- Main RFCs
 - NETCONF Protocol (RFC 6241) <http://www.ietf.org/rfc/rfc6241.txt>
 - YANG Data Modeling Language (RFC 6020) <http://www.ietf.org/rfc/rfc6020.txt>
- Current Work in Progress
 - NETCONF: <http://datatracker.ietf.org/wg/netconf/documents/>
 - NETMOD: <http://datatracker.ietf.org/wg/netmod/documents/>
- WEB Sites
 - NETCONF WG Wiki Page: <http://trac.tools.ietf.org/wg/netconf/trac/wiki>
 - NETCONF Central: <http://www.netconfcentral.org/>
 - YANG Central: <http://www.yang-central.org/>

Part 2

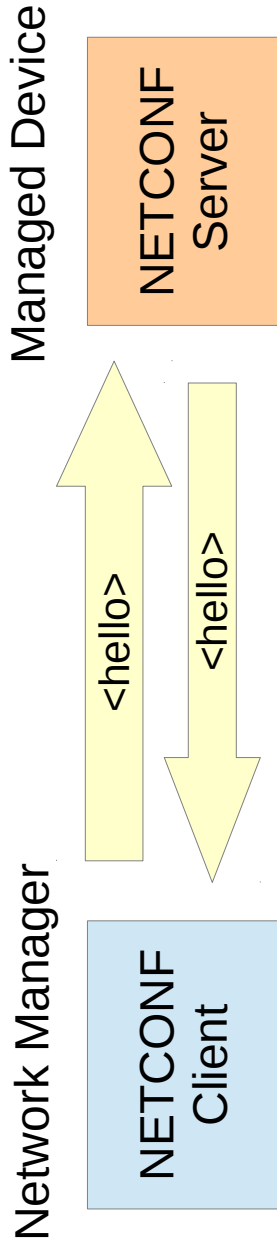
- Overview
 - Comparison of SNMP/SMIPv2 and NETCONF/YANG properties
- NETCONF Basics
 - Overview of core NETCONF protocol concepts
- YANG Basics
 - Overview of core YANG data modeling concepts
- Advanced Topics
 - Overview of advanced NETCONF and YANG concepts

NETCONF Sessions

- NETCONF is session-based
 - Secure transport required
 - SSH/TCP is mandatory; TLS/TCP is optional
 - Transport must provide a user identity to the server for the session (for authorization purposes)
- Why use sessions?
 - Some procedures require multiple protocol operations
 - Some automatic transaction cleanup done when session is dropped or killed

NETCONF <hello>

- NETCONF features are identified with “capability” URIs
- A session starts with an exchange of capability lists by both peers



- The server advertises all its capabilities, assigns a session ID
- The client advertises the protocol versions it supports
 - All optional server features and supported YANG module information can be discovered by the client

Data Editing Model

- SNMP validation and data activation is varbind-based
 - SET PDU can contain a random mix of variable bindings
 - Agent needs to maintain a lot of partial state and activate rows as they are completed in multiple SET PDUs
 - RowStatus used to allow manager to indicate row is complete
- NETCONF validation and data activation is datastore-based
 - Transaction boundaries are clearly identified
 - Incremental data is applied all-or-none with <commit> operation
 - Datastore-wide referential integrity checks built into YANG
 - Exclusive write access possible via full and partial datastore locking

Data Persistence Model

- SNMP data persistence is row-based
 - StorageType column defines persistence behavior for each row
- NETCONF data persistence is datastore-based
 - Entire configuration datastore saved in NV-storage in implementation-specific manner
 - Optional “startup” capability allows access and manual control over configuration used at the next reboot

Defaults Model

- SNMP has DEFVAL clause
 - Implementation hint to agent developers
 - Manager developers cannot rely on it because it is optional
- NETCONF has default statement
 - Mandatory to implement for server developers
 - If the leaf does not exist in the configuration datastore, the default value MUST be used instead

Configuration Defaults

- SNMP has row-based default handling
 - Existence of column representing a default value controlled by the RowStatus column in the same row
- NETCONF has 3 types of default handling
 - Controlled by server and advertised in the “with-defaults” capability URI
 - Normal retrieval of the configuration will not have any leaf that the server treats as a default leaf
 - **trim mode**: if the value matches the YANG default value, the server treats the leaf as a default leaf, and will not create it in the configuration datastore
 - **explicit mode**: if the value is explicitly set by the client then it is stored, even if it matches the YANG default value.
 - **report-all**: the server stores all leaves in the configuration and does not remove any default leaves from the configuration datastore

Server Actions

- SMIV2

- Actions are modeled with MIB objects

```
radiusAccServConfigReset
OBJECT-TYPE
SYNTAX INTEGER
{ other(1),
  reset(2),
  initializing(3),
  running(4) }
MAX-ACCESS read-write
STATUS current
DESCRIPTION
  "Status/action object..."
 ::= { radiusAccServ 4 }
```

- YANG

- All protocol operations including actions are defined in YANG

```
rpc radiusAccServConfigReset {
  description "...";
}
leaf server-state {
  config false;
  type enumeration { ... }
}
```

- There could be a data object defined to monitor the status of the initialization, but:
 - read-only status object
 - not dual purpose

NETCONF Datastores

- There are 3 standard configuration datastores, used in combinations to support different transaction and persistence behavior within the server
 - **running**: mandatory representation of the current running configuration
 - **candidate**: optional scratchpad used to collect edits to apply all-or-none
 - **startup**: optional non-volatile storage representation of the configuration that will be used at the next reboot.
- There are 4 common combinations or editing models identified with capabilities
 - direct (:writable-running), direct + startup (:writable-running + :startup)
 - indirect (:candidate), indirect + startup (:candidate + :startup)

Direct Editing Models

:writable-running



- Edits take effect immediately and are automatically saved to NV-storage

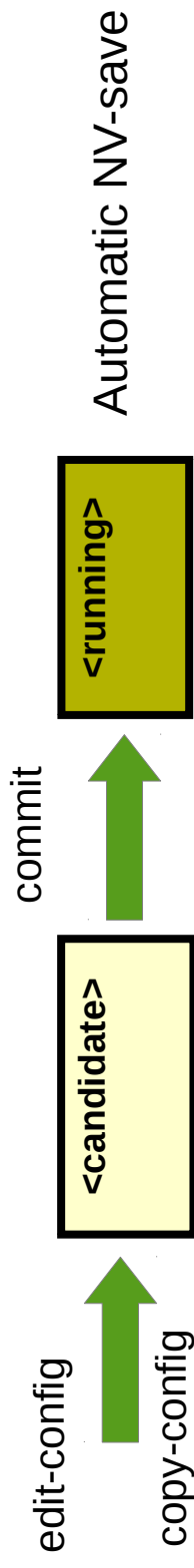
:writable-running + :startup



- Edits take effect immediately and are manually saved to NV-storage

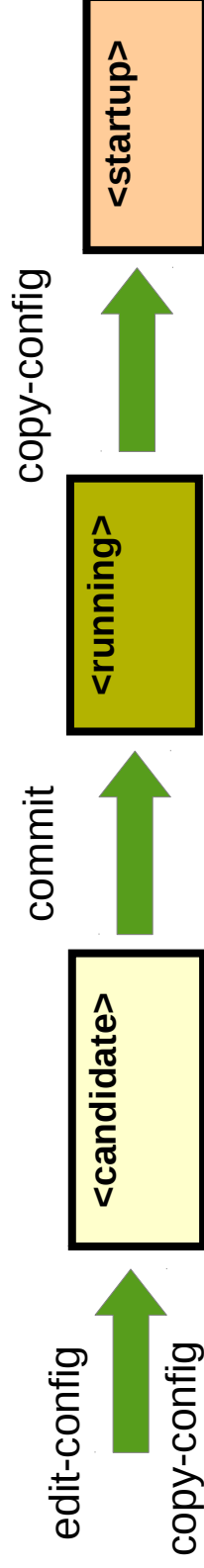
Indirect Editing Models

:candidate



- Edits are collected then applied all-or-none, and automatically saved to NV-storage

:candidate + :startup



- Edits are collected then applied all-or-none, and manually saved to NV-storage


Other Editing Models

- :writable-running + :candidate
 - Generally not supported because there is no standard mechanism to re-sync the candidate to the current configuration and resolve collisions if the running datastore is edited independently of the candidate.
- Private “candidate”
 - Vendor-specific candidate that is not shared by all clients (like the standard candidate datastore)
 - Has same re-sync and collision issues as concurrent editing of running and candidate datastores

NETCONF Message Encoding

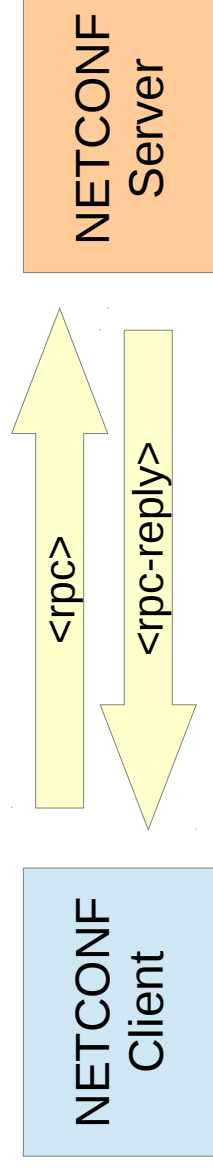
- The “namespace” statement value in the YANG module is used as the XML namespace for an element or qualified attribute
- The YANG identifier is used as the local-name

```
module example-mod {  
  namespace  
    "http://example.com/ns/example-mod";  
  prefix "ex";  
  rpc test { }  
}  
  
<rpc message-id="101"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <test xmlns="http://example.com/ns/example-mod" />  
</rpc>
```

An arrow points from the namespace statement in the YANG module to the xmlns attribute in the XML element.

Remote Procedure Call (RPC)

- Each NETCONF message must be a valid XML instance document
- All operations use a NETCONF specific RPC mechanism
- SNMP does not have a SET PDU that is allowed to return data



- A mandatory “message-id” attribute must be included in an `<rpc>` element
- All operations are defined with the YANG “rpc” statement

<rpc> Input Parameters


- The “input” statement in the “rpc” definition specifies any input parameters for the protocol operation
- Each data-def statement within the input statement is encoded as a child node of the method node. The “input” node is not encoded.

```
module example-mod {  
  namespace  
  "http://example.com/ns/example-mod";  
  prefix "ex";  
  rpc test {  
    input {  
      leaf count { type int32; }  
      leaf msg { type string; }  
    }  
  }  
}  
  
<rpc message-id="101"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <test xmlns="http://example.com/ns/example-mod" >  
    <count>42</count>  
    <msg>first test</msg>  
  </test>  
</rpc>
```

<rpc-reply> Without Output

- If no “output” statement is in the “rpc” definition , then an <ok> element is returned if the operation succeeds.

```
module example-mod {
  namespace
    "http://example.com/ns/example-mod";
  prefix "ex";
  rpc test {
    input {
      leaf count { type uint32; }
      leaf msg { type string; }
    }
    // no output section!!
  }
}
```



<rpc-reply> Output Parameters

- The “output” statement in the “rpc” definition specifies any output parameters for the protocol operation
- Each data-def statement within the output statement is encoded as a child node of the “rpc-reply” node. The “output” node is not encoded.

```
module example-mod {  
  namespace  
  "http://example.com/ns/example-mod";  
  prefix "ex";  
  rpc test {  
    input {  
      leaf count { type uint32; }  
      leaf msg { type string; }  
    }  
    output {  
      leaf memory-errors {  
        type uint32;  
      }  
    }  
  }  
}
```

The diagram illustrates the encoding of the YANG output parameter 'memory-errors' into an XML element. An arrow points from the 'memory-errors' leaf in the YANG model to the corresponding XML element in the output. The XML output is as follows:

```
<rpc-reply message-id="101"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <memory-errors  
    xmlns="http://example.com/ns/example-mod" >  
    3  
  </memory-errors>  
</rpc-reply>
```

<rpc-reply> for Errors

- If any errors occur, then the server is expected to return 1 or more <rpc-error> child nodes within the <rpc-reply> element
- The server is allowed to include “output” data in addition to <rpc-error> nodes
 - But <ok> and <rpc-error> cannot both be returned!
- The server is allowed to include <rpc-error> elements as descendant nodes within output data (e.g., back-end instrumentation retrieval failed)

```
<rpc-reply message-id="101"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <rpc-error>  
    ... contents explained later!!!  
  </rpc-error>  
</rpc-reply>
```

Base Protocol Operations

Name	Description
get-config	Retrieve some or all of a config datastore
edit-config	Edit some or all of a config datastore
copy-config	Copy contents from/to a config datastore
delete-config	Remove all contents of a config datastore
lock	Start exclusive write access of a datastore
unlock	Stop exclusive write access of a datastore
get	Retrieve config and/or state data
close-session	Cause your session to close
kill-session	Force another session to close

<edit-config>

- 5 edit operations (create, merge, replace, delete, remove)

Parameter	Description
target	Configuration datastore to edit
default-operation	Default edit mode; Default (merge) Values (merge, replace, none)
test-option	Values (test-then-set, set, test-only); D: (set)
error-option	Values (stop-on-error, continue-on-error, rollback-on-error); Default (stop-on-error)
config	Portion of the configuration to edit


<edit-config> Options

- **test-option**
 - “test-then-set” causes full validate on candidate
 - “set” will skip the full validation on candidate
 - These 2 values have no affect on the running datastore; always full validation
 - “test-only” will cause the request to be validated but not executed
 - dry-run mode; no data will be altered
- **error-option**
 - very implementation-specific since no processing order is defined for NETCONF edit operations within 1 <edit-config> request
 - some servers refuse to do anything except “rollback-on-error” so the datastore is never left in an an unknown or invalid state

Interface Table Example

```
rpc method: edit-config
input parameters from edit-config rpc
container interfaces {
  list interface {
    key name;
    leaf name { type string; }
    leaf mtu { type uint32; }
  }
}

<rpc message-id="101">
  <edit-config>
    <target><running/></target>
    <default-operation>none</default-operation>
  </config>
  <interfaces>
    <interface>
      <name>eth0</name>
      <mtu operation="replace">1500</mtu>
    </interface>
  </interfaces>
</config>
</edit-config>
</rpc>
```



Set the MTU for Interface ‘eth0’

Using the candidate datastore

- Edit phase
 - <edit-config> used to collect edits in the (shared scratchpad) candidate datastore
 - Global <lock> and <unlock> should be used to prevent multi-client collisions
- Commit phase
 - <commit> used to apply scratchpad all-or-none to the running datastore
- Commit with rollback
 - <commit> w/ <confirmed> parameter for automatic rollback unless confirming <commit> sent by client
 - <cancel-commit> used to force manual rollback
- Cancel an edit
 - <discard-changes> used to remove any edits from the candidate and re-sync with the running datastore

Using the running datastore

- Edit phase
 - <edit-config> used directly on the running datastore
 - Global <lock> and <unlock> should be used to prevent multi-client collisions
 - Subtree-specific <partial-lock> and <partial-unlock> may be supported by the server
- Commit phase
 - There is no commit phase; The edits take effect immediately
- Commit with rollback
 - The server may support the “rollback-on-error” <error-option> to support all-or-none
- Cancel an edit
 - There is no way to cancel an edit

Part 3

- Overview
 - Comparison of SNMP/SMIv2 and NETCONF/YANG properties
- NETCONF Basics
 - Overview of core NETCONF protocol concepts
- YANG Basics
 - Overview of core YANG data modeling concepts
- Advanced Topics
 - Overview of advanced NETCONF and YANG concepts

Basic YANG Module Structure

```
module interfaces {
  yang-version 1;
  namespace "some-unique-URI";
  prefix if;

  import foo-module { prefix foo; }
  include bar-submodule;

  organization "your organization/company name here";
  contact "your name and email address here";
  description "Module summary here";
  reference "Module references here";

  revision 2008-03-05 { description "Initial version."; }

  <data model definitions here, in any order>
}
```

bold = mandatory
plain = optional
black = keywords
green = your text

```
module-stmt
module module-name {
  yang-version 1;
  namespace "uri-string";
  prefix "prefix-string";
  // zero or more imports
  import "module-name" {
    prefix "prefix-string";
    revision-date "date-string";
  }
  // zero or more includes
  include "submodule-name" {
    revision-date "date-string";
  }
  organization "org-name";
  contact "contact info";
  description "module description";
  reference "module reference";
  // zero or more revision identifiers
  revision "date-string" {
    description "revision description";
    reference "revision reference";
  }
  body-stmt; // zero or more
}
```

YANG Statements

- All statements have the same structure
 - statement = keyword [argument] (";" / "{" *statement "}")
 - any statement can terminate (";") or start a new section of sub-statements ("{" ... "}")
- YANG allows external statements to be defined
 - All standard statements begin with a keyword without a prefix
`container`
 - All external statements begin with a keyword with a prefix
`acme:container`
 - The prefix is for the module containing the “extension” statement that defines the external statement
 - All YANG tools are required to skip over an unsupported external statement

YANG Comments

- SMIV2:
 - 2 dashes (--) used to indicate rest of line is a comment
 - Implementation of the Ethernet History group is optional.
 - Consult the MODULE-COMPLIANCE macro for the authoritative conformance information for this MIB.
- YANG:
 - 2 forward slashes (//) used to indicate rest of line is a comment
 - // Implementation of the Ethernet History group is optional.
 - // Consult the MODULE-COMPLIANCE macro for the authoritative conformance information for this MIB.
 - C style multi-line comments also allowed
 - /* Implementation of the Ethernet History group is optional.
Consult the MODULE-COMPLIANCE macro for the authoritative conformance information for this MIB.
*/

YANG String Encoding

- Strings can be specified in several ways:
 - » **Double Quoted String**: whitespace allowed, but it is adjusted (escape character sequences replaced, whitespace trimmed)
 - » **Single Quoted String**: whitespace allowed, and is preserved (this form is safest for pattern strings)
 - » **Unquoted String**: no whitespace allowed (Language tokens like '{' and '}' are not allowed.)
- Strings can be specified in fragments if desired:
 - » “foobarbaz” is the same as “foo” + “bar” + “baz”
 - » Mixing forms is allowed: “foo” + 'bar' + 'baz'

Identifiers in YANG Modules

Built-in definitions

```
type int32;
```

Built-in keywords and type names are not imported.
No prefix is allowed when they are used.

From the current module

```
type fooType; OR type foo:fooType
```

Local definitions such as type names and groupings are not imported.
The current module prefix is allowed, but not required.

From a different submodule

```
include footypes;  
type fooType; OR type foo:fooType
```

An include statement must be present.
The current module prefix is allowed, but not required.

From a different module

```
import bartypes { prefix bar; }  
type bar:barType;
```

An import statement must be present, with a unique prefix.
The declared module prefix must be present when identifiers are used.

YANG Module Revisions

- Every YANG module has a revision
 - most recent revision statement used (date string)
`revision 2014-07-01 {
 description "Initial release";
 reference "RFC 6241";
}`
 - If there are no revision statements, then the module has the 'NULL' revision
 - Only first version allowed! Must add a revision statement to update the module!
- An explicit revision of an external module can be imported

```
import foo {  
  prefix foo;  
  revision-date 2014-07-01;  
}
```

YANG Submodules

- A YANG module can be split up into multiple files, called “submodules”

- Similar syntax to module

```
submodule example-foo-sub1 {  
    belongs-to example-foo { prefix foo; }  
    ....  
}
```

- Each submodule contributes definitions to the module namespace
- The “include” statement is used within the module or a submodule to import definitions from another submodule
- Definitions in the module are not visible in any submodule

YANG Filenames

- SMIv2: no standard naming conventions for MIB modules
- YANG: File naming conventions mandatory for standard modules
 - Directory path information is not part of the specification
 - Organization SHOULD prefix module name
 - 2 forms of a YANG file name
 - any revision:
 - <module-name>.yang
[example-foo.yang](#)
 - specific revision:
 - <module-name>@<revision-date>.yang
[example-foo@2014-06-15.yang](#)

Schema Nodes vs. Data Nodes

- YANG has 2 types of path identifiers

- **schema:** An “augment” statement path can include conceptual nodes

```
augment /nc:get-config/nc:input {
  description
    "Adds the <with-defaults> parameter to the input of the
    NETCONF <get-config> operation.";
  reference "RFC 6243; Section 4.5.1";
  uses with-defaults-parameters;
}
```

- **data:** The “must”, “when”, and “path” statements reference the data tree, which does not contain any conceptual nodes

```
leaf outgoing-interface {
  type leafref { path "../..../rt:interfaces/rt:interface/rt:name"; }
  description
    "Name of the outgoing interface.
    Only interfaces configured for the ancestor routing instance can be given.";
}
```

YANG Body Statements

- **extension**: external statement definition
- **feature**: YANG feature definition
- **identity**: YANG identity definition
- **typedef**: reusable simple type definition
- **grouping**: reusable set of data definitions
- **augment**: add data definitions to another data definition (same or other module)
- **rpc**: protocol operation definition
- **notification**: notification event definition

- All body statements are exported except “augment”

```
body-stmt
[data-def-stmt]
extension extension-name {
  argument argument-name {
    yin-element true|false;
  }
}
feature feature-name {
  if-feature feature-name; // zero or more
  status current|deprecated|obsolete;
  description "description text";
  reference "reference text";
}
data-def-stmt | case-def-stmt; // one or more
rpc rpc-name {
  if-feature feature-name; // zero or more
  status current|deprecated|obsolete;
  description "description text";
  reference "reference text";
  input {
    typedef-stmt; // zero or more
    grouping-stmt; // zero or more
    data-def-stmt; // one or more
  }
  output {
    typedef-stmt; // zero or more
    grouping-stmt; // zero or more
    data-def-stmt; // one or more
  }
}
notification notification-name {
  if-feature feature-name; // zero or more
  status current|deprecated|obsolete;
  description "description text";
  reference "reference text";
  typedef-stmt; // zero or more
  grouping-stmt; // zero or more
  data-def-stmt; // zero or more
}
```

YANG Identities

- Used to declare enumerated values with distributed naming authority
- SMIV2: “OBJECT IDENTITY” macro

- Declares an OBJECT IDENTIFIER value

```
transportDomainUdpIpv4 OBJECT-IDENTITY
STATUS    current
DESCRIPTION
    "The UDP over IPv4 transport domain. ..."
 ::= { transportDomains 1 }
```

- YANG: “identity” statement
 - Declares an identityref leaf or leaf-list value
 - Identity names are scoped by the module containing the identity statement

```
identity transportDomainUdpIpv4 {
    status current
    description
        "The UDP over IPv4 transport domain. ...";
}
```

YANG Features

- All definitions in a YANG module are mandatory to implement, unless they are part of a YANG feature
 - **feature-stmt**: used to declare a YANG feature

```
feature radius {  
  description  
    "Indicates that the device can be configured as a RADIUS client.";  
  reference  
    "RFC 2865: Remote Authentication Dial In User Service (RADIUS)";  
}
```
- SNMP conformance specified separately in MIB module
 - GROUP macro (to specify objects in a named group)
 - MANDATORY-GROUP sub-clause of MODULE-COMPLIANCE macro

YANG Features (2)

- A data definition that includes “if-feature” sub-statements is conditional on server support of all the specified features

- **if-feature-stmt**: used to make a data definition conditional on a YANG feature

```
container radius {  
  if-feature radius;  
  .... // entire 'radius' container is optional to implement  
}
```

- Multiple if-feature statements are combined with logical-AND (all features must be supported)

```
feature radius-authentication {  
  if-feature radius;  
  if-feature authentication;  
  ....  
}
```

YANG Types

- The “type” statement is used to define the data type for a leaf or leaf-list node

```
type uint32 {  
    range “1 .. 100”;  
}
```
- The “typedef” statement is used to define a reusable data type for a leaf or leaf-list node

```
typedef my-index {  
    type uint32 {  
        range “1 .. 100”;  
    }  
}
```

- An in-line “type” and reusable “typedef” are equivalent and can be exchanged in module updates

YANG Numeric Data Types

Numeric content for leaf and leaf-list objects

YANG Type	bits	sign	XSD Type	SMI Type
int8	8	Y	byte	Integer32
uint8	8	N	unsignedByte	Unsigned32
int16	16	Y	short	Integer32
uint16	16	N	unsignedShort	Unsigned32
int32	32	Y	int	Integer32
uint32	32	N	unsignedInt	Unsigned32
int64	64	Y	long	N/A
uint64	64	N	unsignedLong	Counter64
decimal64	64	Y	fixed point	N/A

YANG String Types

String content for leaf and leaf-list objects

YANG Type	XSD Type	SMI Type
string	string	OCTET STRING
binary	base64	OCTET STRING
enumeration	string	INTEGER
bits	list	BITS
instance-identifier	string	OCTET STRING

YANG Special Types

XML content for leaf and leaf-list objects

YANG Type	XSD Type	SMI Type
empty	no type	N/A
boolean	boolean	Integer32
leafref	string	foreign leaf
union	union	N/A
identityref	string	IDENTITY

Data Definition Statements

- container
- leaf
- leaf-list
- list
- choice
- case
- anyxml
- uses

data-def-stmt

```
container name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  must "xpath-expr"; // zero or more  
  presence "presence description";  
  config true/false;  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
  typedef-stmt; // zero or more  
  grouping-stmt; // zero or more  
  data-def-stmt; // zero or more  
}
```

```
leaf name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  type-stmt;  
  units "units name";  
  must "xpath-expr"; // zero or more  
  default "default value";  
  config true/false;  
  mandatory true/false; // no default if true  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
}
```

```
leaf-list name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  type-stmt;  
  units "units name";  
  must "xpath-expr"; // zero or more  
  config true/false;  
  min-elements number; // default zero  
  max-elements number | unbounded;  
  ordered-by user | system;  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
}
```

```
list name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  must "xpath-expr"; // zero or more  
  key "key leafs"; // req. if config=true  
  unique "unique-leafs"; // zero or more  
  config true/false;  
  min-elements number; // default zero  
  max-elements number | unbounded;  
  ordered-by user | system;  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
  typedef-stmt; // zero or more  
  grouping-stmt; // zero or more  
  data-def-stmt; // one or more  
}
```

```
choice name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  default "case name";  
  config true/false;  
  mandatory true/false;  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
  case-stmt | short-case-stmt; // zero or more  
}
```

```
case name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
  data-def-stmt; // zero or more  
}
```

```
anyxml name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  must "xpath-expr"; // zero or more  
  config true/false;  
  mandatory true/false;  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
}
```

```
uses grouping-name {  
  when "xpath-expr";  
  if-feature feature-name; // zero or more  
  status current|deprecated|obsolete;  
  description "description text";  
  reference "reference text";  
  refine-stmt; / zero or more  
  uses-augment-stmt; / zero or more  
}
```

YANG Properties

- Used for validation and referential integrity tests
 - Applies to configuration datastores, <rpc> input, <rpc-reply> output, and <notification> content
- Basic properties:

Name	Type	Purpose
config	boolean	Identify configuration datastore nodes; Inherited from parent, default for top is 'true'
mandatory	boolean	Identify mandatory nodes; default is false
status	enum	Identify definition life-cycle status (same as SMIv2)
must	XPath	Define a validation condition for a node
when	XPath	Define a conditional existence test for a node

“config” statement

- SMIV2

- YANG

- “MAX-ACCESS” clause

- fooEvents OBJECT-TYPE

- SYNTAX Counter32

- MAX-ACCESS read-only

- STATUS current

- DESCRIPTION “...”

- ::= { fooStatsEntry 3 }



- leaf fooEvents {

- type yang:counter32;

- config false;

- status current;

- description “...”;

- }

- fooRetryInterval OBJECT-TYPE

- SYNTAX Integer

- UNITS “Seconds”

- MAX-ACCESS read-create

- STATUS current

- DESCRIPTION “...”

- DEFAULT { 30 }

- ::= { fooControlEntry 2 }



- leaf fooRetryInterval {

- type int32;

- units seconds;

- config true;

- status current;

- description “...”;

- default 30;

- }

“mandatory” statement

- SMIV2
 - DESCRIPTION clause used to define mandatory data node
 - Not the same as the MANDATORY-GROUPS clause, which is used for module compliance
- YANG
 - mandatory “true”:
 - <rpc> input: parameter node MUST be present
 - <notification>: node MUST be present unless filtering and access control causes deletion of node
 - Configuration data: node MUST be present in a valid configuration datastore
 - Non-configuration data: node MUST be present in an <rpc-reply> for <get>, <get-config> unless filtering and access control causes omission of node

“must” statement

- SMIV2: DESCRIPTION clause
- YANG: XPath boolean expression that must be true if the node containing the must-stmt exists; edit rejected if false
 - If within a configuration data node, then the expression can only reference other configuration objects
 - Can be used for multi-object referential integrity validation

```
leaf base-interface {  
  type if:interface-ref;  
  must "/if:interfaces/if:interface[if:name = current()]"  
    + "/vlan:vlan-tagging = 'true'" {  
    description  
      "The base interface must have VLAN tagging enabled.";  
  }  
}
```

“when” statement

- SMIV2: DESCRIPTION clause
- YANG: XPath boolean expression that must be true if the node containing the when-stmt is allowed to exist
 - Sometimes needs to be evaluated even if node does not exist
 - If mandatory=true or min-elements=N statements also present

```
augment "/if:interfaces/if:interface" {  
  when "if:type = 'ianaift:ethernetCsmacd' or  
        if:type = 'ianaift:ieee8023adLag'";  
  leaf vlan-tagging {  
    type boolean;  
    default false;  
  }  
}
```


Part 4

- Overview
 - Comparison of SNMP/SMIPv2 and NETCONF/YANG properties
- NETCONF Basics
 - Overview of core NETCONF protocol concepts
- YANG Basics
 - Overview of core YANG data modeling concepts
- Advanced Topics
 - Overview of advanced NETCONF and YANG concepts

NETCONF Error Reporting

- `<rpc-error>` Fields
- `<rpc-error>` Example
- `<error-tag>` Values
- Data-model specific error information
 - `error-message` and `error-app-tag` statements

<rpc-error> Fields

Name	Description
error-type	Layer associated with the error info (transport, rpc, protocol, application)
error-tag	String identifying the error condition
error-severity	Warning or Error
error-app-tag	Model or application specific error ID
error-path	Path of the element causing the error
error-message	Human-readable error explanation text
error-info	Extended application-level error data

<rpc-error> Example

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path xmlns:t="http://example.com/schema/1.2/config">
      /t:top/t:interface[t:name="Ethernet0/0"]/t:mtu
    </error-path>
    <error-message xml:lang="en">
      MTU value 25000 is not within range 256..9192
    </error-message>
  </rpc-error>
</rpc-reply>
```

<error-tag> Values

error-tag	Description
in-use	The request requires a resource that already is in use
invalid-value	The request specifies an unacceptable value for one or more parameters
too-big	The request or response (that would be generated) is too large for the implementation to handle
missing-attribute	An expected attribute is missing
bad-attribute	An attribute value is not correct; e.g., wrong type, out of range, pattern mismatch
unknown-attribute	An unexpected attribute is present
missing-element	An expected element is missing
bad-element	An element value is not correct; e.g., wrong type, out of range, pattern mismatch
unknown-element	An unexpected element is present
unknown-namespace	An unexpected namespace is present
access-denied	Access to the requested protocol operation or data model is denied because authorization failed

<error-tag> Values (2)

error-tag	Description
lock-denied	Access to the requested lock is denied because the lock is currently held by another entity
resource-denied	Request could not be completed because of insufficient resources
rollback-failed	Request to roll back some configuration change (via rollback-on-error or <discard-changes> operations) was not completed for some reason
data-exists	Request could not be completed because the relevant data model content already exists. For example, a 'create' operation was attempted on data that already exists
data-missing	Request could not be completed because the relevant data model content does not exist. For example, a 'delete' operation was attempted on data that does not exist
operation-not-supported	Request could not be completed because the requested operation is not supported by this implementation
operation-failed	Request could not be completed because the requested operation failed for some reason not covered by any other error condition
malformed-message	A message could not be handled because it failed to be parsed correctly. For example, the message is not well-formed XML or it uses an invalid character set

“error-message” Statement

- The “error-message” statement specifies the <error-message> field in <rpc-error> reply

```
must ". = 0 or not(..../next-hop/weight = 0)" {
  error-message "Illegal combination of zero and non-zero "
    + "next-hop weights.";
  description
    "Next-hop weights must be either all zero (equal
    load-balancing) or all non-zero.";
}
```
- Allowed in specific constraint statements
 - must, range, length, pattern

“error-app-tag” Statement

- The “error-app-tag” statement specifies the <error-app-tag> field in <rpc-error> reply

```
type string {  
    length "0..4";  
    pattern "[0-9a-fA-F]*" {  
        error-app-tag "invalid-tag-identifier";  
    }  
}
```

- Allowed in specific constraint statements
 - must, range, length, pattern
- Standard and vendor-specific values do not work together
 - The ad-hoc list of standard values must override any vendor value
 - String is not qualified so name collisions can occur

Standard <error-app-tag> Values

error-app-tag	error-tag	error-info; error condition
data-not-unique	operation-failed	<non-unique> contains instance identifier; unique-stmt error
data-not-unique	operation-failed	none; <get-schema> duplicate revisions
too-many-elements	operation-failed	none; max-elements error
too-few-elements	operation-failed	none; min-elements error
must-violation	operation-failed	none; must-stmt error
instance-required	data-missing	none; require-instance in instance-identifier or data that does not match leafref type
missing-choice	data-missing	<missing-choice> contains name; mandatory choice is not present
missing-instance	bad-attribute	none; invalid insert operation

“type” Statement

- The “type” statement allows various sub-statements depending on the context
 - Numeric types: **range**
 - Fixed point numeric type 'decimal64'
 - String types: **length** and **pattern**
- Special data types
 - enumeration, bits
 - leafref
 - identityref
 - union

“range” statement

- Numeric types can have a “range” statement like SMIv2

- SMIv2:

```
historyControlBucketsRequested OBJECT-TYPE  
SYNTAX Integer32 (1..65535)
```

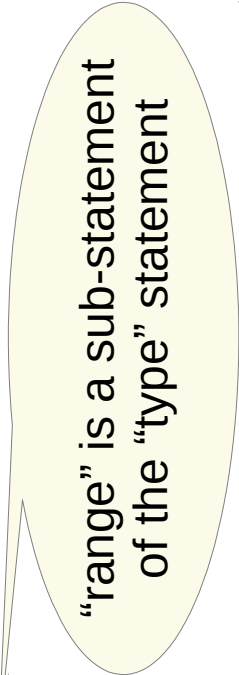
...

- YANG:

```
leaf historyControlBucketsRequested {  
  type int32 { range “1 .. 65535”; }  
  ...  
}
```

...

```
leaf error-range {  
  type int32 { range “1 .. 100 | 1000..2000”; }  
  ...  
}
```



“range” is a sub-statement
of the “type” statement

decimal64 Data Type

- SMIV2: TEXTUAL-CONVENTION for OCTET STRING
- YANG: special data type for 64-bit fixed point integer
 - $i \times 10^n$ where i is an integer64 and n is an integer between 1 and 18
 - **range** and **default** statements are allowed to have fixed-point number
 - **fraction-digits** statement is the number of digits after decimal point

```
typedef my-decimal {  
    type decimal64 {  
        fraction-digits 2;  
        range "1 .. 3.14 | 10 | 20..max";  
    }  
}
```

“length” statement

- String types can have a “length” statement like SMIv2 (SIZE)

- SMIv2:

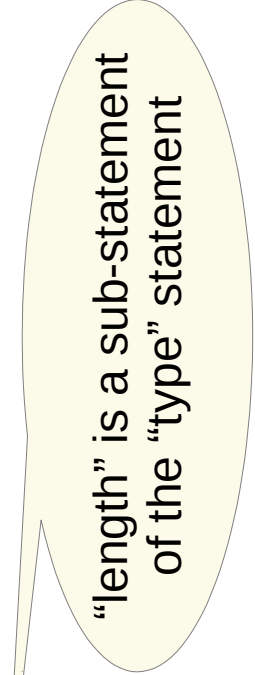
`DateAndTime ::= TEXTUAL-CONVENTION`

...

`SYNTAX OCTET STRING (SIZE (8 | 11))`

- YANG:

```
typedef DateAndTime {  
  type string { length "8 | 11"; }  
  ...  
}
```



“length” is a sub-statement
of the “type” statement

“min” and “max” keywords

- The “range” and “length” statements can use “min” and “max” keywords instead of numeric constants
 - SMIPv2
 - TimeInterval ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
"A period of time, measured in units of 0.01 seconds."
SYNTAX INTEGER (0..2147483647)
 - YANG
 - typedef TimeInterval {
status current;
description
"A period of time, measured in units of 0.01 seconds.";
type int32 { range "0..max"; }
}



Max. for the type can
be replaced with “max”

“min” and “max” keywords (2)

- The “min” and “max” properties are inherited in type and typedef definitions:

- ```
typedef knob-range {
 type int32 { range “0..100”; }
}
```

```
typedef knob2-range {
 type knob-range { range “10 .. max”; }
}
```

```
leaf knob2 {
 type knob2-range;
}
```



“max” for “knob2-range”  
is “100”, not “2147483647”

# “pattern” statement

---

- SMIV2: DESCRIPTION clause
- YANG: regular expression specifying acceptable values for a string leaf or leaf-list value
  - Multiple patterns form logical-AND expression (all patterns must match)
  - Single quotes strings should be used instead of double quotes to prevent unintended translation of special characters

```
type string {
 length "1..max";
 pattern '[a-zA-Z_][a-zA-Z0-9\-_\.]*';
 pattern '^[^xX]*.[^mM]*.[^lL]*';
}
```



# “enumeration” type

---

- SMIv2: enumerated INTEGER type
  - integer value sent on the wire in SNMP data
- YANG: enumeration type
  - **enum** identifier string sent on the wire in NETCONF data
  - identifier allowed to be almost any non-zero length string, except no leading or trailing whitespace

```
type enumeration {
 enum running {
 description "The <running> datastore has changed.";
 value 0; // default auto-assigned value as well
 }
 enum startup {
 description "The <startup> datastore has changed";
 value 1; // default auto-assigned value as well
 }
}
```

# “bits” type

---

- SMIV2: BITS type
  - integer bitmask value sent on the wire in SNMP data
- YANG: bits type
  - list of bit identifier strings sent on the wire in NETCONF data
  - identifier must be a valid YANG identifier string (no whitespace allowed)

```
type bits {
 bit netconf1.0 {
 description "RFC 4741 base:1.0";
 position 0;
 }
 bit netconf1.1 {
 description "RFC 6241 base:1.1";
 position 1;
 }
}
```

# “leafref” type

---

- SMIV2: DESCRIPTION clause; also remote INDEX object
- YANG: **leafref** data type contains the value of another leaf
  - value has the same type as the referenced leaf
  - value is picked from 1 of the existing instances of the referenced leaf
  - Circular references not allowed across leafref nodes
  - **path** is XPath node-set expression identifying subset of referenced leaf instance values allowed to be used

```
leaf mgmt-interface {
 type leafref {
 path "../interface/name";
 }
}
```

# “identityref” type

---

- SMIV2: OBJECT IDENTIFIER data type
  - points at ad-hoc set of OBJECT IDENTITY macros
- YANG: identityref data type can hold an identity value with the same 'base' value.

```
import "crypto-base" { prefix "crypto"; }
identity aes { base "crypto:crypto-alg"; }

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

# “union” type

---

- SMIv2: not supported
- YANG: union data type allows a leaf or leaf-list value to contain 1 of N different data types
  - A value is valid if it is valid for any of the types specified in the union
  - The types are evaluated in order and the data given the first valid type

```
typedef ip-address {
 type union {
 type inet:ipv4-address;
 type inet:ipv6-address;
 }
 description
 "The ip-address type represents an IP address and is IP version neutral.
 The format of the textual representation implies the IP version. This type
 supports scoped addresses by allowing zone identifiers in the address format."
 reference
 "RFC 4007: IPv6 Scoped Address Architecture";
}
```

# YANG Data Definition Statements

---

- Used to define simple or complex data structures
  - leaf
  - leaf-list
    - ordered-by
  - container
    - presence
  - list
    - key
    - unique
    - min-elements
    - max-elements
  - choice
    - case

# “leaf” statement

---

- SMIV2: “OBJECT-TYPE” macro for a scalar or column
- YANG: used to represent a singular node containing a simple type

```
leaf name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 type-stmt;
 units “units name”;
 must “xpath-expr”; // zero or more
 default “default value”;
 config true/false;
 mandatory true/false; // no default if true
 status currentdeprecatedobsolete;
 description “description text”;
 reference “reference text”;
}
```

# “leaf-list” statement

---

- SMIv2: no comparable language construct available
- YANG: used to represent a list of nodes containing a simple type; leaf-list value is considered the key

```
leaf-list name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 type-stmt;
 units “units name”;
 must “xpath-expr”; // zero or more
 config true|false;
 min-elements number; // default zero
 max-elements number | unbounded;
 ordered-by user | system;
 status current|deprecated|obsolete;
 description “description text”;
 reference “reference text”;
}
```



# “ordered-by” statement

---

- SMIv2: use a table with an INDEX component to specify the order
- YANG: list and leaf-list instances can either be unordered (ordered-by system) or client-ordered (ordered-by user)
- If a leaf-list or list is user-ordered, then the YANG “insert” extensions to the NETCONF <edit-config> operation can be used
  - insertion mode can be 1 of:
    - first
    - last (default)
    - before or after an existing instance

# “container” statement

---

- SMIv2: no comparable language construct available
- YANG: used to represent a singular node containing zero or more child subtrees

```
container name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 must “xpath-expr”; // zero or more
 presence “presence description”;
 config true|false;
 status current|deprecated|obsolete;
 description “description text”;
 reference “reference text”;
 typedef-stmt; // zero or more
 grouping-stmt; // zero or more
 data-def-stmt; // zero or more
}
```

# “presence” statement

---

- SMIv2: no comparable language construct available
- YANG: 2 types of containers in configuration datastores
  - “presence container”, also called “P container”:
    - The “presence” statement assigns a meaning to the presence of a container in the data tree. It takes as an argument a string that contains a textual description of what the node's presence means.

```
container ntp {
 if-feature ntp;
 presence
 "Enables the NTP client unless the 'enabled' leaf
 (which defaults to 'true') is set to 'false'";
 description
 "Configuration of the NTP client.";

}
```

# NP-container

---

Container without a presence container is a “non-presence container”, also called “NP-container”

- Only used for structuring data
  - Transparent to some constraints
    - **mandatory**: If the NP-container contains mandatory child nodes then the NP-container itself is mandatory
    - **default**: If the NP-container contains and default child nodes then the NP-container itself and the default child nodes are considered present
  - Not transparent to other constraints:
    - **must** and **when** statements apply the same to P and NP containers
  - Server MAY delete empty NP containers
    - Client should not create empty NP containers anyway

# “list” statement

---

- SMIV2: “OBJECT-TYPE” macro for a table entry
- YANG: used to represent a list of nodes containing a complex type

```
list name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 must “xpath-expr”; // zero or more
 key “key leafs”; // req. if config=true
 unique “unique-leafs”; // zero or more
 config true|false;
 min-elements number; // default zero
 max-elements number | unbounded;
 ordered-by user | system;
 status current|deprecated|obsolete;
 description “description text”;
 reference “reference text”;
 typedef-stmt; // zero or more
 grouping-stmt; // zero or more
 data-def-stmt; // one or more
}
```

# “key” statement

---

- SMIv2: INDEX clause
- YANG: used to identify an instance of a list
  - key is ordered sequence of 1 or more leafs
  - keys must be child leafs of list; encoded first in XML, not YANG order
  - key is mandatory if config=true, optional if config=false
  - keys are accessible in NETCONF
  - mandatory=true is redundant for a key leaf

```
list server {
 key "ip port";
 leaf ip { ... }
 leaf port { ... }
 ...
}
```

# “unique” statement

---

- SMV2: DESCRIPTION clause
- YANG: used to specify uniqueness requirements across all instances of a list object
  - **unique:** The “unique” statement is used to put constraints on valid list entries. It takes as an argument a string that contains a space-separated list of schema node identifiers, which MUST be given in the descendant form (see the rule “descendant-schema-nodeid” in Section 12). Each such schema node identifier MUST refer to a leaf.

```
list server {
 key "name";
 unique "ip port";
 leaf name { type string; }
 leaf ip { type inet:ip-address; }
 leaf port { type inet:port-number; }
}
```

–

# “min-elements” statement

---

- SMIv2: DESCRIPTION clause
- YANG: used to specify minimum number of instances of a list object
  - same application to <rpc>, <rpc-reply>, <notification>, and configuration data as the mandatory-stmt
  - default is zero

```
list endpoint {
 key address;
 min-elements 1;
 ordered-by user;
 leaf address { ... }
}
```

–



# “max-elements” statement

---

- SMIv2: DESCRIPTION clause
- YANG: used to specify maximum number of instances of a list object
  - same application to <rpc>, <rpc-reply>, <notification>, and configuration data as the mandatory-stmt
  - 2 forms of statement
    - `max-elements 100; // max-elements <positive integer>;`
    - `max-elements unbounded;`
  - default is “unbounded”

# “choice” statement

---

- SMIV2: DESCRIPTION clauses across OBJECT-TYPE macros
- YANG: used to specify a conceptual selection point for zero or one of N possible cases
  - **mandatory** “true”: can be used to require 1 case to be present
  - **default** “case-A”: can be used to select a default case
  - A choice cannot have both “default” and “mandatory” statements

```
choice name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 default “case name”;
 config true/false;
 mandatory true/false;
 status currentdeprecatedobsolete;
 description “description text”;
 reference “reference text”;
 case-stmt | short-case-stmt; // zero or more
}
```

# “case” statement

---

- SMIV2: DESCRIPTION clauses across OBJECT-TYPE macros
- YANG: used to specify 1 selection within a choice of N possible cases
  - default case cannot contain any mandatory objects
  - **short form**: a data-def statement can be used directly as if a case-stmt with the same name was declared

```
choice A {
 case one {
 leaf one { type string; }
 }
}

choice B { // same as choice A
 leaf one { type string; }
}
```

```
case name {
 when “xpath-expr”;
 if-feature feature-name; // zero or more
 status current|deprecated|obsolete;
 description “description text”;
 reference “reference text”;
 data-def-stmt; // zero or more
}
```

# Choice in Operation

---

- The choice and case names (and layers in the object tree) are NOT present in any instance document conforming to the YANG schema (names are somewhat transparent)
  - A choice name cannot conflict within any of its siblings in the parent data object, or any top-level object in the same module if it is a top-level choice
  - A case name cannot conflict within any of its siblings in the choice
  - A child object name within a case cannot conflict with any of its siblings in the case, or any child nodes of any other case within the choice, or any sibling of the choice statement
- If a new case is created then the existed case is automatically deleted by the server (even in candidate)

# YANG Reuse Statements

---

- Any module can augment another module (no loops)
  - augment
    - SPARSE AUGMENTS example
  - uses
    - refine

# “augment” statement

---

- SMIv2: AUGMENTS clause in OBJECT-TYPE for entry

- ifXEntry OBJECT-TYPE

....

```
AUGMENTS { ifEntry }
```

```
::= { ifXTable 1 }
```

- YANG: augment statement

- no mandatory objects allowed
  - 3 forms of augment-stmt
    - top-level, for same-module
    - top-level, for another module
    - within a “uses” statement, to augment an expansion of a grouping always for same module

```
augment “/path/to/augment/target” {
 when “when-xpath-expr”;
 if-feature feature-name; // zero or more
 status current|deprecated|obsolete;
 description “description text”;
 reference “reference text”;
 data-def-stmt | case-def-stmt; // one or more
}
```

# Sparse AUGMENTS

---

- SMIV2: “DESCRIPTION” clause in table entry used to explain the conditions when the augmenting columns will be added to the augmented table entry
- YANG: “when” statement used within “augment” statement to define the conditions when the augmenting data will be added to the augmented data

```
augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
 when "../rt:address-family = 'v6ur:ipv6-unicast'" {
 description
 "This augment is valid only for IPv6 unicast.";
 }
 description
 "This leaf augments an IPv6 unicast route.";
 leaf destination-prefix {
 type inet:ipv6-prefix;
 description
 "IPv6 destination prefix.";
 }
}
```

# “uses” statement

---

- SMIv2: not supported
- YANG: **uses** statement expands a YANG grouping
  - The namespace of nodes in the grouping assigned to ‘final’ expansion within a real data definition
  - Can be used within a grouping or within a data-def-stmt
  - **refine** and **augment** sub-statements used to customize the grouping expansion
  - **must**, **when**, and **path** expressions are evaluated in a special way
    - Nodes with prefixes are bound to a namespace in the module where they are defined (i.e., the grouping module)
    - Nodes without prefixes are bound to the namespace of the final expansion within a real data-def-stmt

```
container global-lock {
 description "Present if the global lock is set.";
 uses lock-info;
}
```



# “refine” statement

---

- SMIv2: not supported
- YANG: refine statement used to alter sub-statements
  - Specific list of changes allowed defined in RFC 6020, section 7.12.2

```
container http-server {
 leaf name {
 type string;
 }
 uses acme:endpoint {
 refine port {
 default 80;
 }
 }
}
```



default added to the  
'port' leaf definition

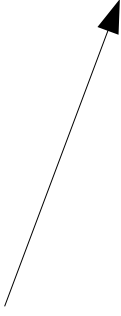
# YANG Extensions

---

- extension statement
  - defines 1 external statement
- prefixed keyword indicates an external statement

```
extension default-deny-all {
 description
 "Used to indicate that
 the data model node
 controls a very sensitive
 security system
 parameter. ...";
}

leaf shared-secret {
 type string;
 mandatory true;
 nacm:default-deny-all;
 description
 "The shared secret which is
 known to both the RADIUS
 client and server.";
 reference
 "RFC 2865: Remote
 Authentication
 Dial In User Service";
}
```



# NETCONF Notifications

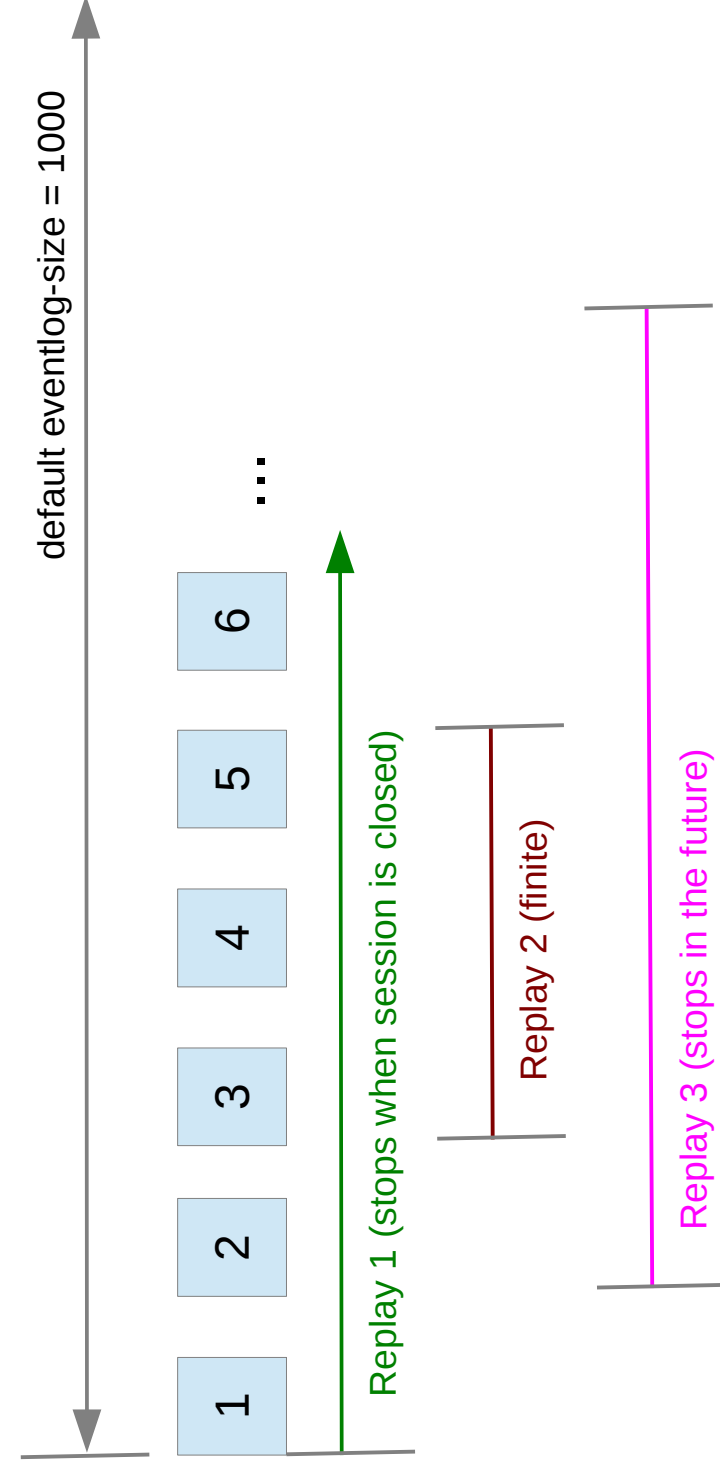
---

- Requires the client to maintain a NETCONF session
  - Client sends <create-subscription> operation to start receiving notification messages
    - subscribes to 1 specific event stream
    - filter can be provided to select specific events, event data
    - start and stop times can be provided
  - Server maintains 1 or more notification event streams
    - “NETCONF” is the only standard stream name at this time
    - A replay buffer is maintained by the server, sorted by timestamp
      - A “replay request” from the client will cause stored events to be sent to the client for the specified time interval
      - start and stop times can be in the past or future, to control when collection begins and ends

# Notification Replay Buffer

---

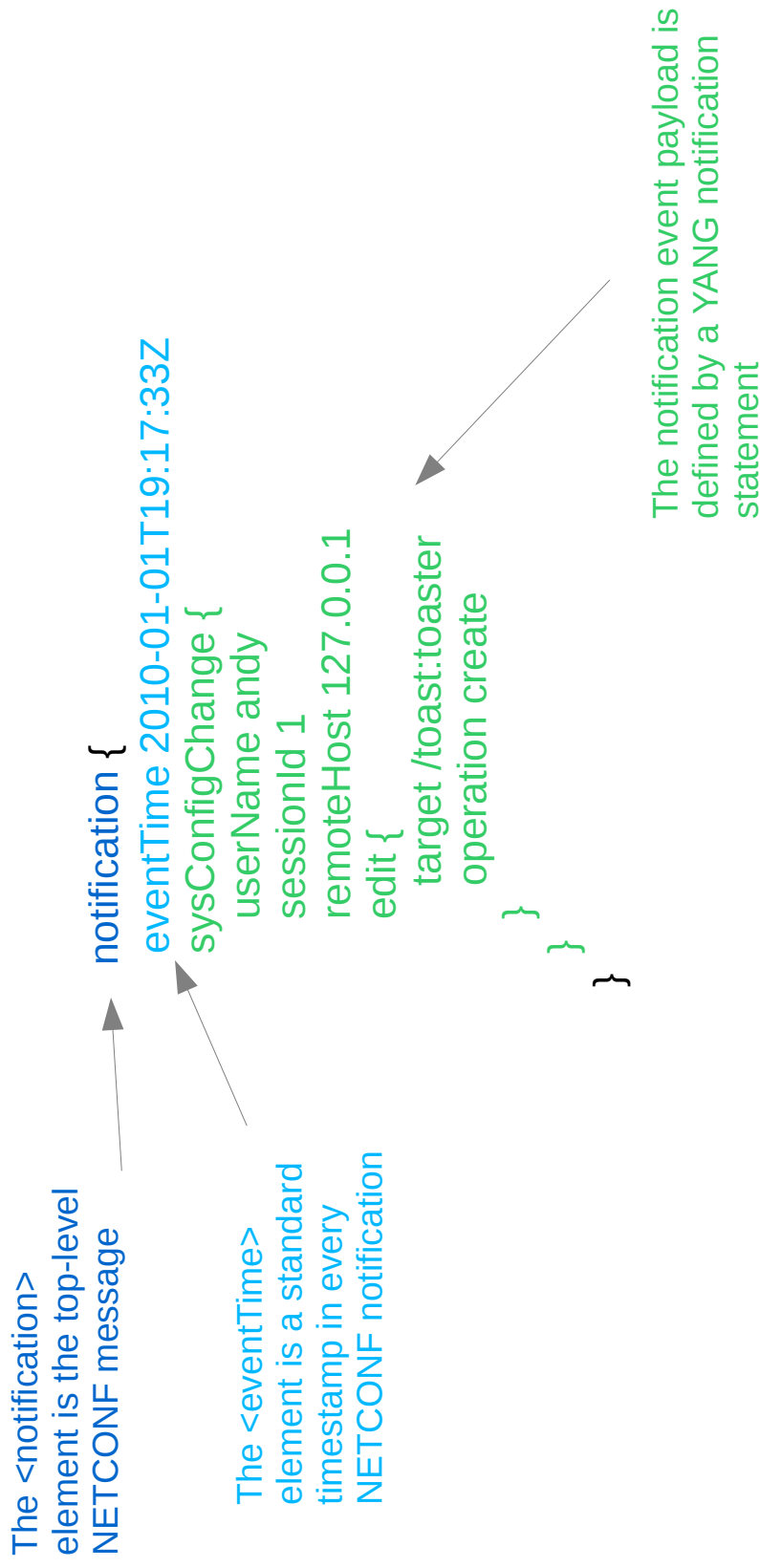
## System Event Log



Replay 1: startTime=2009-01-01, no stopTime  
Replay 2: startTime=2009-12-01, stopTime=2009-12-31  
Replay 3: startTime=2009-10-01, stopTime=2011-12-31

# Notification Structure

---



# Future Topics

---

- Topics not covered in this tutorial

# Data Retrieval (TBD-1)

---

- TBD: explain MIB walk with GET and GETBULK vs. <get-config> or <get> with filters (subtree and XPath)
- TBD: explain how custom RPC is used for hardwired filters
- TBD: show trade-off between duplication of INDEX in varbind list (not duplicated in NETCONF) vs. XML duplication of identifier name hierarchy
- TBD: XPath needle-in-a-haystack example
- TBD: XPath returns full tree w/ keys filled in, not node-set

# Advanced Concepts (TBD-2)

---

- Local groupings and typedefs
  - nested in data tree; cannot override any parent with the same name
- Nested grouping expansion
- YANG module update rules
  - cannot break old clients
- MIN-ACCESS vs. deviation-stmt
- NETCONF session management
  - close-session, kill-session, auto-cleanup



# Advanced Concepts (TBD-3)

---

- Subtree filtering
- XPath filtering
- YANG “deviation” statement
- Confirmed commit procedure
- Network-wide commits
  - roll forward or roll backward using confirmed commit procedure when executing edit across multiple NETCONF devices (all-or-none)
- Access Control Model: NACM vs. VACM

# NETCONF Standards (TBD-4)

---

- TBD: list all RFCs
- TBD: slide explaining
  - ietf-netconf-monitoring
  - ietf-partial-lock
  - ietf-netconf-notifications
  - ietf-with-defaults
    - report-all and report-all-tagged mode to force retrieval of defaults
  - ietf-interfaces
- TBD: standards work in progress

# Summary

---

- **NETCONF (RFC 6241)**
  - Supports request, response, and notification messages
  - Modular content defined with YANG data modeling language
  - Focus on configuration management operations
  - Extensible operations and error reports
- **YANG (RFC 6020)**
  - Supports powerful data modeling constructs like choices, containers, and nested lists
  - Uses XPath and other machine-readable validation mechanisms
  - Supports data reuse and language extensions