NETCONFとYANGの話

浅間正和@有限会社銀座堂

NETCONF/YANGとは？

- NETCONF
  - "NETwork CONFiguration Protocol" の略
  - ネットワーク機器の設定を投入したり削除したりするためのプロトコル
  - 他にもネットワーク機器の統計情報などを取得したりネットワーク機器からの通知を受け取ったりいろいろできる
- YANG
  - "Yet Another Next Generation" の略(なんのこっちゃｗ)
  - NETCONFでサーバーとクライアントがデータのやり取りをする際に「どういったデータをやり取りするのか？」を記述するための言語

NETCONF/YANGとは？

- NETCONF
  - "NETwork CONFiguration Protocol" の略
  - ネットワーク機器の設定を投入したり設定を削除したりするためのプロトコル
  - 他にもネットワーク機器の統計情報などを取得したりネットワーク機器からの通知を受け取ったりいろいろできる
- YANG
  - "Yet Another Next Generation"の略(なんのこっちゃｗ)
  - NETCONFでサーバーとクライアントがデータのやり取りをする際に「どういったデータをやり取りするのか？」を記述するための言語

ん？

NetworkをManagementするProtocol？

SNMPは？

SNMP/SMIv2とNETCONF/YANGの違い

	SNMP/SMIv2	NETCONF/YANG
トランスポート層	UDP	TCP (SSHやTLS)
トランザクション のサポート	なし	あり (コミットとロールバック)
データフォーマット	独自TLV	XML
設定変更	set	edit-config
設定情報や 統計情報の取得	get	get get-config
ネットワーク機器 からのイベント通知	trap	notification
ネットワーク機器 への操作		RPCを定義可能

YANGの起源

6. YANG Syntax

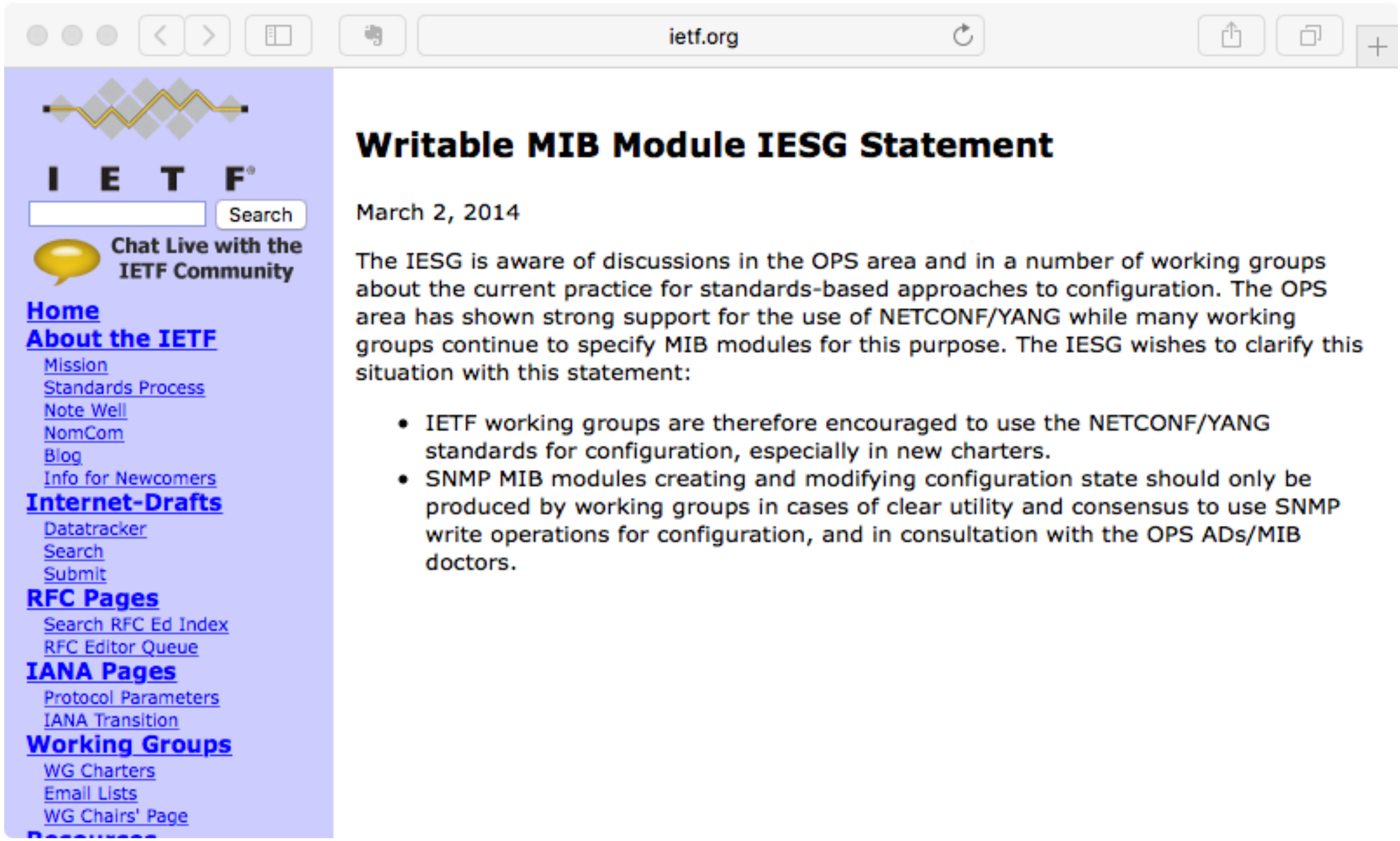
The YANG syntax is similar to that of SMIng [RFC3780] and programming languages like C and C++. This C-like syntax was chosen specifically for its readability, since YANG values the time and effort of the readers of models above those of modules writers and YANG tool-chain developers. This section introduces the YANG syntax.

RFC7950 "The YANG 1.1 Data Modeling Language" から引用

The SMIv1 and the SMIv2 are bound to the Simple Network Management Protocol (SNMP) [RFC3411], while the SPPI is bound to the Common Open Policy Service Provisioning (COPS-PR) Protocol [RFC3084]. Even though the languages have common rules, it is hard to use common data definitions with both protocols. It is the purpose of this document to define a common data definition language, named SMIng, that can formally specify data models independent of specific protocols and applications. The appendix of this document defines a core module that supplies common SMIng definitions.

RFC3780 "SMIng - Next Generation Structure of Management Information" から引用

SNMP/SMIv2からNETCONF/YANGへ



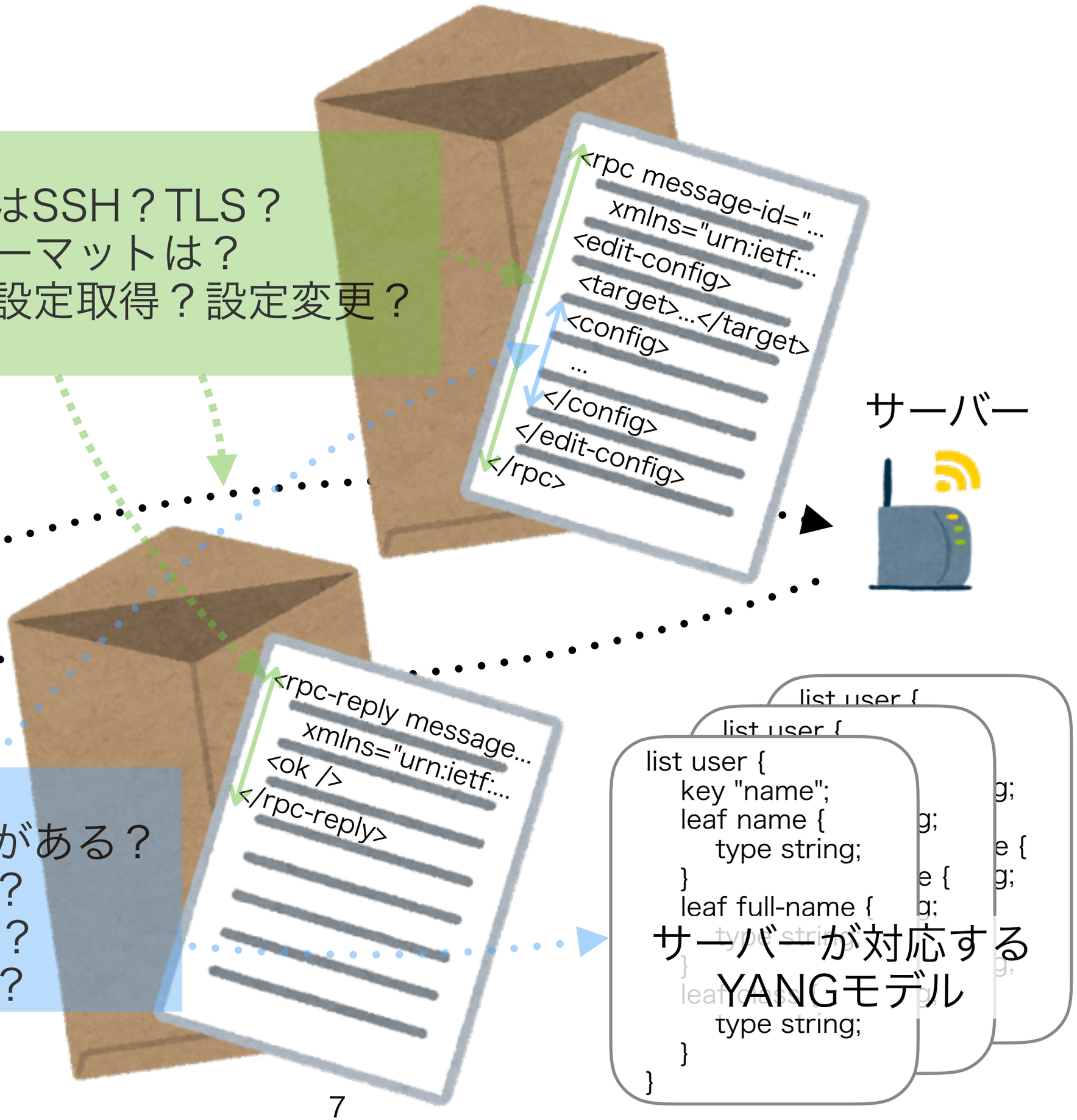
https://www.ietf.org/iesg/statement/writable-mib-module.html

ざっくり

NETCONF
・トランスポート層はSSH？TLS？
・メッセージのフォーマットは？
・オペレーションは設定取得？設定変更？
・エラーの時は？

クライアント

サーバー



YANG
・どんなパラメータがある？
・パラメータの型は？
・どんなRPCがある？
・どんな通知がある？

イラスト©いらすとや

NETCONF

NETCONFのメッセージ

- <hello>
  - 接続した直後に交わされる対応するケーパビリティの交換
- <rpc>
  - クライアントからサーバーへ送られるRPCの内容を表す
  - get/get-config/edit-configなどのNETCONFの基本オペレーション(後述)やYANGモデルで定義されたRPCの発行
  - クライアントから送られるメッセージのほとんどはこれ
- <rpc-reply>
  - サーバーからクライアントへ送られるRPCの結果を表す
  - RPCが成功したかやエラーがあった場合はその内容を含む
- <notification>
  - 非同期でやり取りされる通知メッセージ
  - 購読した通知が発生した時に送られる

<hello>による対応ケーパビリティの交換

- NETCONFのセッションが確立された際に<hello>メッセージでお互いのケーパビリティが交換される
- ケーパビリティはURIで表される

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>
            urn:ietf:params:netconf:base:1.1
        </capability>
        <capability>
            urn:ietf:params:netconf:capability:startup:1.0
        </capability>
        <capability>
            http://example.net/router/2.3/myfeature
        </capability>
    </capabilities>
    <session-id>4</session-id>
</hello>
```

<hello>による対応ケーパビリティの交換

- Writable-Running Capability
  - urn:ietf:params:netconf:capability:writable-running:1.0
  - デバイスが稼働中(Running)の設定を直接変更できる機能
  - (後述する)edit-config等のターゲットにrunningを指定できる
- Candidate Configuration Capability
  - urn:ietf:params:netconf:capability:candidate:1.0
  - デバイスが候補(Candidate)の設定を持つことができる機能
  - commitオペレーションとdiscard-changesオペレーションが可能
- Confirmed Commit Capability
  - urn:ietf:params:netconf:capability:confirmed-commit:1.1
  - 候補(Candidate)の設定を反映(Commit)する前に確認手順を踏むことができる機能
- Rollback-on-Error Capability
  - urn:ietf:params:netconf:capability:rollback-on-error:1.0
  - エラーが発生した際にロールバックすることができる機能

<hello>による対応ケーパビリティの交換

- Validate Capability
  - urn:ietf:params:netconf:capability:validate:1.1
  - 設定にエラーがないか検証することができる機能
- Distinct Startup Capability
  - urn:ietf:params:netconf:capability:startup:1.0
  - 起動時用の設定を持つことができる機能
- URL Capability
  - urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,...
  - copy-config(後述)等で対象をURLで指定することができる機能
- XPath Capability
  - urn:ietf:params:netconf:capability:xpath:1.0
  - getやget-configなど(後述)の対象の指定(filter)にXPath表現を用いることができる機能

<rpc>と<rpc-reply>

- クライアントからサーバーに対してRPCを呼び出す際にクライアントからサーバーに対して<rpc>メッセージが送信される
- サーバーからクライアントに対してRPCの呼び出し結果が送信される際にサーバーからクライアントに対して<rpc-reply>メッセージが送信される
- <rpc>には必ずmessage-idというアトリビュートをつけなければならず対応する<rpc-reply>の識別に用いる
- RPCの呼び出しの際に何かしらのエラーが発生したときは<rpc-error>が返される(RFC7803のAppendix A参照)

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:ex="http://example.net/content/1.0"
  ex:user-id="fred">
    <get/>
</rpc>
```

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:ex="http://example.net/content/1.0"
  ex:user-id="fred">
    <data>
        <!-- contents here... -->
    </data>
</rpc-reply>
```

NETCONFの基本オペレーション(1)

- <get>
  - 稼働中の設定(running config)とデバイスの状態を取得
- <get-config>
  - 指定されたデータストアの設定の全てまたは一部を取得
- <edit-config>
  - 指定されたデータストアの設定の全てまたは一部を変更
- <copy-config>
  - 設定をコピーする
  - 設定はデータストアをまたがることもできる
- <delete-config>
  - 設定を削除する
  - ただし稼働中の設定(running config)は削除できない

NETCONFの基本オペレーション(2)

- <lock>
  - 排他制御のために設定データストア全体をロックする
- <unlock>
  - ロックを解放する
- <close-session>
  - NETCONFセッションを終了する
- <kill-session>
  - 他のNETCONFセッションを終了させる

NETCONFの追加オペレーション

- <commit>
  - 候補(Candidate)の設定を稼働中(Running)の設定に反映
  - Candidate Configuration Capabilityを持つ時に利用可能
- <discard-changes>
  - 候補(Candidate)の設定を破棄
  - Candidate Configuration Capabilityを持つ時に利用可能
- <cancel-commit>
  - 候補(Candidate)の設定を反映する前の確認で反映をキャンセルする
  - Confirmed Commit Capabilityを持つ時に利用可能
- <validate>
  - 候補(Candidate)の設定を検証する
  - Validate Capabilityを持つ時に利用可能

YANG

YANGのバージョン

- バージョン1
  - RFC6020(2010年10月)
- バージョン1.1
  - RFC7950(2016年8月)
- 基本的には同じだが結構微妙な変更がある
  - " でクオートされた文字列のエスケープが微妙に違う
  - クオートされていない文字列に ' と " を入れれない
  - いくつかのステートメントでサブステートメントが可になったり不可になったり
  - 新しいサブステートメントも追加されていたり
    - 詳細は RFC7950 の "1.1. Summary of Changes from RFC 6020" を参照
    - 1 と 1.1 で違う時は明記するようにします

はじめてのYANG

- キーワード [引数] ";"
  あるいは
  キーワード [引数] "{"
  　　*サブステートメント
  "}"
  をステートメントと呼ぶ
- ステートメントによっては
  引数部分にステートメント
  の名前を表す識別子を持つ
- YANGモデルはmoduleという
  ステートメントで表現される
  (他にsubmoduleもある(後述))

```
module example-system {
        yang-version 1.1;
        namespace "urn:example:system";
        prefix "sys";
        organization "Example Inc.";
        contact "joe@example.com";
        description
                "The module for entities
                 implementing the Example system.";
revision 2007-06-09 {
        description "Initial revision.";
}
container system {
        leaf host-name {
                type string;
                description
                "Hostname for this system.";
        }
}
...
}
```

はじめてのYANG

moduleステートメント

識別子

containerステートメント
(moduleステートメントの
サブステートメント)

leafステートメント
(containerステートメントの
サブステートメント)

```
module example-system {
    yang-version 1.1;
    namespace "urn:example:system";
    prefix "sys";
    organization "Example Inc.";
    contact "joe@example.com";
    description
          "The module for entities
           implementing the Example system.";
    revision 2007-06-09 {
        description "Initial revision.";
    }
    container system {
        leaf host-name {
            type string;
            description
            "Hostname for this system.";
        }
    }
    ...
}
```

moduleステートメント

- 識別子にモジュール名を指定する
- 以下のサブステートメントを持つ
  - ヘッダ情報
    - yang-version、namespace、prefix
  - リンケージステートメント
    - import、include
  - メタ情報
    - organization、contact、description、reference
- リビジョンヒストリ
  - revision
- モジュール定義 ← これが本体！

moduleのサブステートメント

- yang-versionステートメント
  - YANGモデルを記述するYANGのバージョンを記す
  - 1の時はオプションだったが1.1からは必須になった
- organizationステートメント
  - YANGモデルを管理する組織の情報を記す
- contactステートメント
  - YANGモデルを管理する組織への問い合わせ先を記す
- descriptionステートメント
  - YANGモデルの説明を記す
- referenceステートメント
  - 参照情報を記す(RFCなど)
- revisionステートメント
  - 改訂した日付を記す

namespaceとprefix

- YANGではすでに定義された別のYANGモデルを参照することができる(具体的な方法は後述)
- 定義した識別子が他のYANGモデルのものと被っても大丈夫なよう名前空間とそれを参照するための接頭辞を定義できる
- namescapeステートメント
  - YANGモデルの名前空間を記述する
  - 名前空間はグローバルに一意なURI(RFC3986)
    - 例) urn:example:system、http://example.jp/system
- prefixステートメント
  - 他のYANGモデルがこのYANGモデルの識別子を参照する際につける接頭辞
    - 例) sys(他のYANGモデルがこのYANGモデルの識別子idを参照する場合sys:idのように参照できる)

importによる外部YANGモデル参照

- importステートメントを用いることで外部のYANGモデルを参照することができる

```
module a {
        yang-version 1.1;
        namespace "urn:example:a";
        prefix "a";
        revision 2008-01-01 { ... }
        grouping a {
                leaf eh { ... }
        }
}
```

```
module b {
        yang-version 1.1;
        namespace "urn:example:b";
        prefix "b";
        import a {
                prefix "p";
                revision-date 2008-01-01;
        }
        container bee {
                uses p:a;
        }
}
```

- importする際にrevision-dateサブステートメントを用いることで特定のリビジョンのYANGモデルを指定できる

includeとsubmodule

- 大きなYANGモデルはsubmoduleに分割しmoduleから includeで呼び出すことでまとめることができる

```
module example-system {
        yang-version 1.1;
        namespace "urn:example:system";
        prefix "sys";
        include example-types;
        organization "Example Inc.";
        contact "joe@example.com";
        description
          "The module for entities implementing
           the Example system.";
        revision 2007-06-09 {
                description "Initial revision.";
        }
        // definitions follow...
}
```

```
submodule example-types {
        yang-version 1.1;
        belongs-to "example-system" {
                prefix "sys";
        }
        organization "Example Inc.";
        contact "joe@example.com";
        description
          "This submodule defines common
           Example types.";
        revision "2007-06-09" {
                description "Initial revision.";
        }
        // definitions follow...
}
```

いよいよモジュール定義…でもその前に…

- スキーマ
  - 要するに型(例: integer、string)
- データ
  - 要するに値(例: 123、"abc")
- スキーマノード
  - スキーマツリーを構成するひとつのノード
    - 具体的には以下のステートメントがスキーマノードとなる
      - action(1.1から)、container、leaf、leaf-list、list、choice、case、rpc、input、output、notification、anydata(1.1から)、anyxml
- スキーマツリー
  - モジュールに定義されたスキーマノードの階層構造

いよいよモジュール定義…でもその前に…

- データ定義ステートメント
  - 新しいデータノードを定義するためのステートメント
  - 具体的には以下のステートメント
    - container、leaf、leaf-list、list、choice、case、augment、uses、anydata(1.1から)、anyxml
- データノード
  - データツリーを構成するひとつのノード
  - 具体的には以下のステートメントからデータノードが作られる
    - container、leaf、leaf-list、list、anydata(1.1から)、anyxml
- データツリー
  - NETCONFでやり取りされる階層化されたデータ表現

いよいよモジュール定義…でもその前に…

- 要するに…
  - NETCONFでやり取りされるデータツリーにはcontainer、leaf、leaf-list、list、anydata(1.1から)、anyxmlのデータノードしか載りません
  - それ以外のデータ定義ステートメントは最終的には上記のステートメントに展開されます
- これ以降は以下のように説明を進めます
  - データツリーを構成する上記のステートメントの説明
  - 上記ステートメントの定義で用いるサブステートメントの説明(型、制約、設定のためのノードか否か、等)
  - 上記のステートメント以外のデータ定義ステートメントの説明(choice、case、augment、uses)
  - その他(rpc、action(1.1から)、notification、等)

containerステートメント

- ノードを "格納" するためのノードを表す
- 引数に識別子を取りデータツリーに表現されるときは識別子がXMLの要素として表現される
- 中にはサブステートメントで記述されたデータツリーが格納される

```
container system {
    description
      "Contains various system params.";
    container services {
      description
        "Configure externally available
         services.";
      container "ssh" {
        presence "Enables SSH";
        description
          "SSH service-specific config.";
          // more leafs, containers, and...
      }
    }
}
```

⇒

```
<system>
    <services>
        <ssh/>
    </services>
</system>
```

leafステートメント

- ある値を保持する "葉" となるノードを表す
- 引数に識別子を取りデータツリーに表現されるときは識別子がXMLの要素として表現される
- 中にはサブステートメントで定義された型や制約に応じた文字列が格納される

```
leaf port {
    type inet:port-number;
    default 22;
    description
        "The port to which the SSH server
         listens.";
}
```

⇒ <port>2022</port>

leaf-listステートメント

- leafのように値を保持する "葉" となるノードを表すがleafと異なり複数のデータノードを表すことができる

```
leaf-list allow-user {
    type string;
    description
        "A list of user name patterns to
         allow.";
}
```

⇒

```
<allow-user>alice</allow-user>
<allow-user>bob</allow-user>
```

listステートメント

- containerのようにノードを "格納" するためのノードを表すがcontainerと異なり複数のデータノードを表現することができる
- リスト内のノードを一意に識別するためのleafの一覧をkeyサブステートメントで指定する

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

⇒

```
<server>
    <name>smtp</name>
    <ip>192.0.2.1</ip>
    <port>25</port>
</server>
<server>
    <name>http</name>
    <ip>192.0.2.1</ip>
    <port>25</port>
</server>
```

anydataとanyxmlステートメント

- なんでもありなデータノードを表す
- YANGバージョン1ではanyxmlしか無かったが1.1からanydataも追加された
- anydata
  - YANGでモデル化可能だがモジュールの設計段階ではよく分からないデータに使用
- anyxml
  - よく分からないXMLデータに使用

anyxml html-info; ⇒

```
<html-info>
    <p xmlns="http://www.w3.org/1999/xhtml">
        This is <em>very</em> cool.
    </p>
</html-info>
```

typeステートメント

- leafとleaf-listが持つ値の "型" を指定するステートメント
- 型は組込型(Built-in Types)と派生型(Derived Types)の二種類に分類される
- 組込型として以下の19個が定義されている
  - binary ... バイナリデータを表す型
  - bits ... ビットあるいはフラグのセットを表す型
  - boolean ... 真偽値("true" か "false")を表す型
  - decimal64 ... 64-bit の符号付10進数
  - empty ... 値を持たないleafを表す
  - enumeration ... 列挙型
  - identityref ... 識別子への参照
  - instance-identifier ... データツリーノードへの参照
  - (続く…)

typeステートメント

- (組込型の定義続き)
  - int8、inet16、int32、int64 ... 各ビット幅の符号付き整数型
  - leafref ... リーフへの参照
  - string ... 文字列型
  - uint8、uint16、uint32、uint64 ... 各ビット幅の符号無し整数型
- 派生型は組込型や他の派生型からtypedefステートメントを用いて定義することができる(詳細は後述)
- typeステートメントはその型に応じて補足するためのサブステートメントを指定する場合がある(詳細は後述)

typedefステートメント

- 組込型や派生型から派生型を定義するためのステートメント
- 定義した派生型はtypeステートメントでleafやleaf-listの型として用いることができる
- typedefで派生型を定義する際デフォルトの値や単位を指定することができる

```
typedef percent {
    type uint8 {
        range "0 .. 100";
    }
}

leaf completed {
    type percent;
}
```

⇒ <completed>20</completed>

lengthとpatternステートメント

- 型がstring組込型とbinary組込型あるいはそれらの派生型の時にその長さ(length)と正規表現による制約(pattern)を指定することができる
- patternを指定する場合modifierサブステートメントに引数invert-matchで意味を反転させることができる(1.1から)
- 右記の例では
  - 文字列の長さが1文字以上
  - アルファベットか_で始まり0文字以上のアルファベットか"_"か数字か"-"か"."が続き
  - 大文字か小文字の"xml"で始まらない

文字列を表す

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

rangeとfraction-digitsステートメント

- 型が数値を表す組込型(int*、uint*、decimal64)あるいはそれらの派生型の時にその範囲(range)を指定することができる
- またdecimal64組込型かその派生型の時に小数点以下の桁数(fraction-digits)を指定することができる
- 右記の例では
  - 小数点以下は2桁で
  - 値が
    - 1から3.14までか
    - 10か
    - 20以上か
- の10進数を表す

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

baseステートメント

- 型がidentityref組込型あるいはその派生型の時にベースとなる識別子を指定することでその識別子をベースに定義された識別子のみを指定することができる
- 右記の例ではリーフcryptoは識別子crypto-algをベースに定義された識別子を指定することができる
- 具体的には識別子des3と識別子aesを指定することができる

<crypto>des3</crypto> ⇐

```
identity crypto-alg {
    description
      "Base identity from which all
       crypto algorithms are derived.";
}
identity des3 {
    base "crypto-alg";
}
identity aes {
    base "crypto-alg";
}
leaf crypto {
    type identityref {
        base "crypto-alg";
    }
}
```

bitステートメント

- 型がbits組込型あるいはその派生型の時にサブステートメントとしてbitステートメントを定義することでその取りうる値を定義することができる
- bitステートメントにはpositionサブステートメントを指定することでビット列のポジションを指定することができる
  - 0から始まるposition番目のビットが1になる

```
typedef mybits-type {
    type bits {
        bit disable-nagle { position 0; }
        bit auto-sense-speed { position 1; }
        bit ten-mb-only { position 2; }
    }
}
leaf mybits {
    type mybits-type;
    default "auto-sense-speed";
}
```

⇒

```
<mybits>disable-nagle
                ten-mb-only</mybits>
```

enumステートメント

- 型がenumeration組込型あるいはその派生型の時にサブステートメントとしてenumステートメントを定義することでその取りうる値を定義することができる
- enumステートメントにはvalueサブステートメントを指定することでその値を指定することができる

```
leaf myenum {
    type enumeration {
        enum zero;
        enum one;
        enum seven {
            value 7;
        }
    }
}
```

⇒ <myenum>seven</myenum>

pathステートメント

- 型がleafref組込型あるいはその派生型の時にサブステートメントとしてpathステートメントを定義することで取りうる値を指定することができる
- pathの指定にはXPathを用いる
  - pathで用いることのできる関数などについてはRFC7950の "10. XPath Functions" を参照

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

⇒

```
<interface>
    <name>eth0</name>
</interface>
<interface>
    <name>lo</name>
</interface>
<mgmt-interface>eth0</mgmt-interface>
```

configステートメント

- configステートメントを用いることでそのノードが設定のためのものなのか状態を表すものなのかを指定する
- configステートメントはcontainer、leaf、leaf-list、list、choice、anydata(1.1から)、anyxmlとdeviateステートメントのサブステートメントとして指定することができる
- configが "true" に設定されたノードは<edit-config>等で設定を変更することができる
- configが "false" に設定されたノードは<get>に含まれるが<get-config>には含まれない

presenceステートメント

- データノードを何も持たないcontainerノード(空のcontainerノード)はサーバ側で削除することができる
- ただし空でも意味を持つcontainerノードを定義したい時もある(例えば27ページの識別子sshのcontainerノード)
- このようなcontainerノードの場合presenceステートメントを定義することで空であってもサーバが削除しないように指定することができる
- presenceステートメントの引数にはその説明を渡す
- presenceステートメントはcontainerステートメントのみが持つことができる

mustステートメント

- mustステートメントではデータノード間の制約条件をXPath表現で記述することができる
- mustステートメントはcontainer、leaf、leaf-list、list、anydata(1.1から)、anyxml、input(1.1から)、output(1.1から)、notification(1.1から)とdeviateのサブステートメントとして指定することができる

```
container interface {
    leaf ifType {
        type enumeration {
            enum ethernet;
            enum atm;
        }
    }
    leaf ifMTU {
        type uint32;
    }
    must 'ifType != "ethernet" or ifMTU = 1500' {
        error-message
            "An Ethernet MTU must be 1500";
    }
    must 'ifType != "atm" or'
          + ' (ifMTU <= 17966 and ifMTU >= 64)' {
        error-message
            "An ATM MTU must be 64 .. 17966";
    }
}
```

mandatoryステートメント

- mandatoryステートメントを用いることでそのデータノードが値を持つことが必須か否かを指定することができる
- mandatoryステートメントには "true" か "false" の引数を指定でき "true" の時にそのノードが必須になる
- mandatoryステートメントはleaf、choice、anydata(1.1から)、anyxmlとdeviateステートメントのサブステートメントとして指定することができる

min-elementsとmax-elements

- 複数のノードを持つことができるleaf-listとlistでは制約条件として最小要素数と最大要素数を指定することができる
- min-elementsが指定されていない時のデフォルトは0が指定された時と同じである
- max-elementsが指定されていない時のデフォルトはunbounded(無制限)と同じである
- min-elementsステートメントとmax-elementsステートメントはleaf-list、listとdeviateステートメントのサブステートメントとして指定することができる

ordered-byステートメント

- 複数のノードを持つことができるleaf-listとlistではサーバー内部で保持する際の順序をどのように管理するべきかという問題がある
- ユーザー一覧のように特に順序に意味がないものもあればACLのように順序が変わると意味も変わってしまうようなものもある
- ordered-byステートメントの引数に "system" を指定した場合その順序はシステムに任される(例: ユーザー一覧)
- ordered-byステートメントの引数に "user" を指定した場合順序はユーザーで管理することになりシステムが勝手に順序を入れ替えることはなくなる(例: ACL)
- ordered-byステートメントはleaf-listとlistのサブステートメントとして指定することができる

uniqueステートメント

- list内のleafの値がユニークでなければならない時にuniqueステートメントを用いることでユニークとなるよう制約条件を指定することができる
- uniqueステートメントにはleafの識別子を空白文字区切りで指定でき複数のleafが指定された時はそれらの組み合わせがユニークとなるような制約条件となる
- uniqueステートメントで指定されたleafのデータノードが存在しない時はそのleafは制約条件から外れる
- uniqueステートメントはlistステートメントとdeviateステートメントのサブステートメントとして指定することができる

choiceとcaseステートメント

- 複数の選択肢があるような時にchoiceステートメントとcaseステートメントを用いることでそれらを表現できる
- 下記の例ではprotocolコンテナの中は2パターンの可能性がありudpリーフがデータノードとして存在するときはaのケースでtcpリーフが存在するときはbのケースとなる
- choiceとcaseはデータノードには現れずchoiceの位置にcaseの中のデータ定義ステートメントのデータノードが現れる

```
container protocol {
    choice name {
        case a {
            leaf udp { type empty; }
        }
        case b {
            leaf tcp { type empty; }
        }
    }
}
```

⇒

```
<protocol>
    <tcp/>
</protocol>
```

choiceとcaseステートメント

- caseステートメントに含まれるデータ定義ステートメントの識別子はユニークである必要がある
- 例えば例えば右記の例ではaケースとbケースの両方にethernetという同じ識別子のデータ定義ステートメントが存在するため不正となる

```
// 不正なYANGモデル
choice interface-type {
    case a {
        leaf ethernet { ... }
    }
    case b {
        container ethernet { ...}
    }
}
```

- choiceがひとつのデータ定義ステートメントからなるようなときは下記のような略記が可能である

```
choice interface-type {
    container ethernet { ... }
}
```

=

```
choice interface-type {
    case ethernet {
        container ethernet { ... }
    }
}
```

groupingとusesステートメント

- groupingステートメントを用いることで再利用可能なノードの塊を定義することができる
- groupingステートメントで定義したノードの塊はusesステートメントを用いることで利用することができる
- 下記の例では複数の箇所で頻繁に用いられるIPアドレスとポート番号の組み合わせをendpointという識別子でグループ化しhttp-serverというコンテナの中で使用している

```
grouping endpoint {
    description
          "A reusable endpoint group.";
    leaf ip { type inet:ip-address; }
    leaf port { type inet:port-number; }
}
container http-server {
    leaf name { type string; }
    uses endpoint;
}
```

⇒

```
<http-server>
    <name>extern-web</name>
    <ip>192.0.2.1</ip>
    <port>80</port>
</http-server>
```

groupingとusesステートメント

- usesステートメントでグループ化された塊を展開するときはそこにそれらのデータ定義ステートメントが存在しているのと同じ扱いとなるため識別子が重複するデータノードは定義できない(右記の例)

```
// 不正なYANGモデル
container http-server {
    uses endpoint;
    leaf ip { type string; }
}
```

- usesステートメントのサブステートメントとしてrefineステートメントを用いることでグループ化されたノードに制約条件等のサブステートメントを追加することができる

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

- 後述するaugmentでもカスタマイズできる

augmentステートメント

- augmentステートメントを用いることですでに定義されている別モジュールのYANGモデルやgroupingでグループ化されたノードにスキーマツリーを接ぎ木することができる
- augmentステートメントの引数にはスキーマノード識別子を指定する
- augmentステートメントがモジュールやサブモジュールのトップレベルにあるときは絶対パスでスキーマノードを指定する
- augmentステートメントがusesステートメントのサブステートメントのときは相対パスでスキーマノードを指定する
- whenサブステートメントを用いることで接ぎ木する条件をXPath表現で指定することができる

```
module example-interface-module {
    namespace
        "urn:example:interface-module";
    container interfaces {
        list ifEntry {
            key "ifIndex";
            leaf ifIndex { type uint32; }
            leaf ifDescr { type string; }
            leaf ifType { type iana:IfType; }
            leaf ifMtu { type int32; }
        }
    }
}
module example-module {
    namespace "urn:example:ds0";
    import example-interface-module {
        prefix "if";
    }
    augment "/if:interfaces/if:ifEntry" {
        when "if:ifType='ds0'";
        leaf ds0ChannelNumber {
            type ChannelNumber;
        }
    }
}
```

⇒

```
<interfaces
  xmlns="urn:example:interface-module"
  xmlns:ds0="urn:example:ds0">
    <ifEntry>
        <ifIndex>1</ifIndex>
        <ifDescr>Flintstone Inc ...</ifDescr>
        <ifType>ethernetCsmacd</ifType>
        <ifMtu>1500</ifMtu>
    </ifEntry>
    <ifEntry>
        <ifIndex>2</ifIndex>
        <ifDescr>Flintstone Inc ...</ifDescr>
        <ifType>ds0</ifType>
        <ds0:ds0ChannelNumber>
                1</ds0:ds0ChannelNumber>
    </ifEntry>
</interfaces>
```

rpcステートメント

- rpcステートメントを用いることでNETCONFの基本オペレーションのような独自のオペレーションを定義できる

```
rpc activate-software-image {
    input {
        leaf image-name {
            type string;
        }
    }
    output {
        leaf status {
            type string;
        }
    }
}
```

⇒

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netc...">
    <activate-software-image
    xmlns="http://example.com/system">
        <image-name>
            example-fw-2.3</image-name>
    </activate-software-image>
</rpc>
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netc...">
    <status
    xmlns="http://example.com/system">
        The image example-fw-2.3 is ...
    </status>
</rpc-reply>
```

actionステートメント

- actionステートメントを用いることで特定のコンテナやリストに対して実行するオペレーションを定義できる
- actionステートメントはYANGバージョン1.1から利用可能

```
list interface {
    key "name";
    leaf name { type string; }
    action ping {
        input {
            leaf destination {
                type inet:ip-address;
            }
        }
        output {
            leaf packet-loss {
                type uint8;
            }
        }
    }
}
```

⇒

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netc...">
    <action xmlns="urn:ietf:params:...">
        <interface xmlns="http://exampl...">
            <name>eth1</name>
            <ping>
                <destination>
                    192.0.2.1</destination>
            </ping>
        </interface>
    </action>
</rpc>
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netc..."
  xmlns:sys="http://example.com/syst...">
    <sys:packet-loss>60</sys:packet-loss>
</rpc-reply>
```

notificationステートメント

- notificationステートメントを用いることで通知メッセージでやり取りされるノードのスキーマツリーを定義できる

```
module example-event {
    yang-version 1.1;
    namespace "urn:example:event";
    prefix "ev";
    notification event {
        leaf event-class {
            type string;
        }
        leaf reporting-entity {
            type instance-identifier;
        }
        leaf severity {
            type string;
        }
    }
}
```

⇒

```
<notification
  xmlns="urn:ietf:params:xml:ns:netco...">
    <eventTime>
    2008-07-08T00:01:00Z</eventTime>
    <event xmlns="urn:example:event">
        <event-class>fault</event-class>
        <reporting-entity>
          /ex:interface[ex:name='Ethernet0']
        </reporting-entity>
        <severity>major</severity>
    </event>
</notification>
```

extensionステートメント

- extensionステートメントを用いることで独自のステートメントを定義できる
- extensionステートメントにはargumentサブステートメントを定義できそのそのステートメントを利用する時の引数がargumentで指定した識別子と対応づけられる

```
module example-extensions {
    yang-version 1.1;
    ...
    extension c-define {
        description
            "Takes as an argument a name
             string.
             Makes the code generator use
             the  given name in the #define.";
        argument "name";
    }
}
```

⇒

```
module example-interfaces {
    yang-version 1.1;
    ...
    import example-extensions {
        prefix "myext";
    }
    ...
    container interfaces {
        ...
        myext:c-define "MY_INTERFACES";
    }
}
```

featureとif-featureステートメント

- featureステートメントでフィーチャーの識別子を定義しておき他のステートメントでそのフィーチャーが有効な時にそのステートメントも有効になるような制約を持たせることができる
- featureステートメントはmodule、submoduleステートメントのサブステートメントとして利用することができる
- if-featureステートメントはcontainer、leaf、leaf-list、list、choice、case、anydata(1.1から)、anyxml、uses、rpc、action(1.1から)、notification、augment、identity(1.1から)、feature、enum(1.1から)、bit(1.1から)ステートメントのサブステートメントとして指定することができる

featureとif-featureステートメント

```
module example-syslog {
    feature local-storage {
        description
            "This feature means that the server supports local storage (memory, flash, or
             disk) that can be used to store syslog messages.";
    }
   container syslog {
        leaf local-storage-limit {
            if-feature local-storage;
            type uint64;
            units "kilobyte";
            config false;
            description
                "The amount of local storage that can be used to hold syslog messages.";
        }
    }
}
// if-feature は and や or で結合することもできる。
// if-feature "outbound-tls or outbound-ssh";
// if-feature "not foo or bar and baz";
// if-feature "(not foo) or (bar and baz)";
```

deviationとdeviateステートメント

- YANGモデルで定義された情報が実装と異なるような時にdeviationステートメントを用いることでYANGモデルで定義した情報を書き換えることができる
- deviationステートメントの引数には書き換えたいノードをXPathで指定する
- deviationステートメントのサブステートメントとしてdeviateステートメントを用いることで具体的な書き換え内容を指定する
- deviateの引数には "not-supported"(未対応)、"add"(条件を追加)、"replace"(条件を差し替え)、"delete"(条件を削除)のいずれかを指定する

deviationとdeviateステートメント

```
// daytimeノードはサポートしていない
deviation /base:system/base:daytime {
    deviate not-supported;
}

// typeのデフォルトはadmin
deviation /base:system/base:user/base:type {
    deviate add {
        default "admin";
    }
}
```

```
// name-serverは3つまで指定することが可能
deviation /base:system/base:name-server {
    deviate replace {
        max-elements 3;
    }
}

// mustによる制約条件を削除
deviation /base:system {
    deviate delete {
        must "daytime or time";
    }
}
```

参考情報

- RFC4741: NETCONF Configuration Protocol
  - https://tools.ietf.org/html/rfc4741
- RFC6241: Network Configuration Protocol (NETCONF)
  - https://tools.ietf.org/html/rfc6241
- draft-ietf-netconf-rfc5277bis: Subscribing to Event Notifications
  - https://tools.ietf.org/html/draft-ietf-netconf-rfc5277bis-01
- RFC5277: NETCONF Event Notifications
  - https://tools.ietf.org/html/rfc5277
- RFC7950: The YANG 1.1 Data Modeling Language
  - https://tools.ietf.org/html/rfc7950
- RFC6020: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)
  - https://tools.ietf.org/html/rfc6020

- RFC6991: Common YANG Data Types
  - https://tools.ietf.org/html/rfc6991
- RFC7223: A YANG Data Model for Interface Management
  - https://tools.ietf.org/html/rfc7223
- RFC7277: A YANG Data Model for IP Management
  - https://tools.ietf.org/html/rfc7277
- RFC8022: A YANG Data Model for Routing Management
  - https://tools.ietf.org/html/rfc8022

- draft-ietf-netmod-rfc6087bis: Guidelines for Authors and Reviewers of YANG Data Model Documents
  - https://tools.ietf.org/html/draft-ietf-netmod-rfc6087bis-07