TTC技術レポート
Technical Report

TR-1087

YANG 1.1 データモデリング言語に関する技術報告書

Technical Report on Young Modeling Language

第 1 版
2021 年 4 月 8 日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TTC Telecommunication Technology Committee

目次

＜参考＞

１．国際勧告等の関連

本技術レポートは、RFC7950 を調査したものである。

２．上記国際勧告等に対する追加項目等

なし

３．改版の履歴

版数	制定日	改版内容
第１版	2021 年 4 月 8 日	制定

４．参考文献

[RFC7950] M. Bjorklund, Ed.., " The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

５．工業所有権

本標準に関わる"工業所有権等の実施の権利に係る確認書"の提出状況は、TTC ホームページでご覧になれます。

６．技術レポート作成部門

第１版　：　企業ネットワーク専門委員会

I　本技術レポートの概要

これまで、ネットワークの状態可視化には、広く SNMP (Simple Network Management Protocol) が使用されてきた。しかし、SNMP の最新バージョンである SNMPv3 は 1999 年に登場したレガシーなプロトコルであり、様々な課題を持っている。そのため、IETF では 2001 年頃から継続して SNMP に代わるネットワーク管理プロトコルの議論が行われている。

2001 年当時から言われている SNMP の課題のひとつは、SNMP は装置の状態取得に使われることが多く、装置の設定に関してはサポートされている機能が少ないという点である。例えば、SNMP はトランザクションの仕組みを持っておらず、設定の妥当性の確認やロールバックといったことができない。また、バイナリベースのプロトコルであり、プログラマがあまり親しみのない SNMP 独自のプロトコルを理解しなければならないため、独自の MIB に対して操作を行うことへの敷居がやや高いという課題もあった。

これらの課題を解決するために、IETF では 2002 年頃から SNMP に代わるネットワーク管理プロトコルとして、NETCONF/YANG の標準化が進められてきた。

NETCONF/YANG は、近年の大規模データセンターの普及でネットワークの仮想化が進んだことにより、より柔軟にネットワークを運用管理が可能な技術として、SDN (Software Defined Networking) や NFV (Network Function Virtualization) で利用されるようになった。国内では一部の機器が NETCONF/YANG をサポートしているのみであるが、海外では Cisco や Juniper Networks といった大手ベンダを始めとして、既に多くのルータベンダが NETCONF/YANG をサポートしている。

また、最近の議論では管理対象デバイスから管理装置に YANG データをプッシュする、YANG push の議論も進んでいる。2020 年 2 月現在、YANG push の RFC は一部を除きまだドラフトの状態であるが、改版は進んでおり標準化も近いと予測される。

本報告書では、YANG の最新仕様である RFC7950「The YANG 1.1 Data Modeling Language」のセクション 1 からセクション 18 について、日本語に翻訳する。

II　RFC7950 原文の著作権について

III　RFC7950 の和訳

概要

YANG は、ネットワーク管理プロトコルの設定データ、状態データ、リモート・プロシージャ・コール、および通知をモデル化するために使用されるデータモデリング言語である。この文書は YANG 言語のバージョン 1.1 の構文と意味を記述する。YANG バージョン 1.1 は YANG 言語のメンテナンスリリースで、オリジナル仕様書の曖昧さや欠陥に対処する。YANG バージョン 1 との下位互換性には、少数の非互換性がある。また、この文書は Network Configuration Protocol (NETCONF)への YANG マッピングを規定する。

1. 序文

YANG は、当初、Network Configuration Protocol (NETCONF)、NETCONF Remote Procedure Calls、および NETCONF notifications [RFC6241]によって操作される、設定および状態データをモデル化するために設計されたデータモデリング言語である。YANG バージョン 1[RFC6020]の発行以降、YANG は他のプロトコル(例えば、RESTCONF [RESTCONF]や Constrained Application Protocol (CoAP) Management Interface (CoMI) [CoMI])のために使用されたり、使用されることが提案されたりしてきた。さらに、XML 以外のエンコーディングも提案されている(例: JSON [RFC7951])。

この文書は YANG 言語のバージョン 1.1 の構文と意味を記述する。また、YANG モジュールで定義されたデータモデルが、XML(Extensible Markup Language)[XML]でどのようにエンコードされるか、および、NETCONF の操作を使用してデータを操作する方法についても説明している。他のプロトコルとのエンコードが可能であるかは、この仕様の範囲外である。

YANG データモデルの開発に関して、[YANG-Guidelines]はいくつかのガイドラインと推奨事項を提供する。この文書は RFC 6020 [RFC 6020]を廃止しないことに注意する。

1.1. RFC 6020 からの変更の要約

この文書は YANG 言語のバージョン 1.1 を定義する。YANG バージョン 1.1 は YANG 言語のメンテナンスリリースであり、元の仕様[RFC6020]のあいまいさと欠陥に対処している。

以下の変更点は、YANG バージョン 1 との下位互換性がない:

- ダブルクォーテーションで表される文字列のエスケープ文字の解釈規則を変更した。これは、YANG バージョン 1 との下位互換性のない変更点である。YANG バージョン 1 のモジュールを 1.1 に更新するとき、モジュールが現在無効な文字列を使用している場合、新しいルールに適合するように文字列を変更する必要がある。詳細は、セクション 6.1.3 を参照すること。
- 引用符で囲まれていない文字列には、シングルクォート文字またはダブルクォート文字を含めることはできない。これは、YANG バージョン 1 との下位互換性のない変更点である。YANG バージョン 1 のモジュールを 1.1 に更新するとき、モジュールがこのような引用符文字を使用する場合、新しいルールに適合するように文字列を変更する必要がある。詳細は、セクション 6.1.3 参照する。
- リストキーで"when" と"if-feature" が不正になった。これは、YANG バージョン 1 との下位互換性のない変更点である。YANG バージョン 1 のモジュールを 1.1 に更新するとき、モジュールがこれらの構文を使用する場合、新しいルールに適合するように削除する必要がある。
- YANG モジュールで正当な文字を定義した。YANG バージョン 1 のモジュールを 1.1 に更新する場合、不正になった文字はすべて削除する必要がある。詳細についてはセクション 6 参照のこと。
- 組み込みデータ型の"string" で非文字を不正にした。この変更は、YANG ベースのプロトコルの実行時動作に影響する。

YANG には、以下の追加変更が行われている:

- YANG のバージョンを"1"から"1.1"に変更した。

- YANG バージョン"1.1" で"yang-version" ステートメントが必須になった。
- "if-feature" 構文を任意管理項目名の論理式に拡張した。
- "bit"、"enum"、および"identity"の"if-feature"を許可する。
- "refine"で"if-feature"を許可する。
- "choice" を"case" ステートメントの省略形として許可する(セクション 7.9.2 参照)。
- pattern ステートメントに新しい部分ステートメント"modifier"を追加した(セクション 9.4.6 参照)。
- "input"、"output"、"notification" で"must" を許可する。
- leafref で"require-instance" を許可する。
- "import"と"include"の"description"と"reference"を許可する。
- 複数のリビジョンのモジュールのインポートを許可する。
- 条件付きで必須のノードを追加するために"augment" を許可する(セクション 7.17 参照)。
- セクション 10 に新しい XML Path Language (XPath) 関数のセットを追加した。
- セクション 6.4.1 で XPath コンテキストのツリーを明確にした。
- XPath 式で identityref の文字列値を定義した(セクション 9.10 参照)。
- typedefs の leafref での接頭辞のない名前の意味を明確にした(セクション 6.4.1 と 9.9.2 参照)。
- 同一性を複数の基本同一性から導き出せるようにする(セクション 7.18 とセクション 9.10 参照)。
- enumeration と bits のデータ型についてサブタイプ化を許可する(セクション 9.6 とセクション 9.7 参照)。
- leaf-lists にデフォルト値を持たせる(セクション 7.7.2 参照)。
- 非設定の leaf-lists(セクション 7.7 参照)で一意でない値を許可する。
- 文法で大文字と小文字を区別する文字列([RFC7405]に従う)の構文を使用する。
- module のアドバタイズのメカニズムを変更した(セクション 5.6.4 参照)。
- submodule の定義のスコープ規則を変更した。submodule は、"include" ステートメントを使用せずに、同じ module に属するすべての submodule のすべての定義参照できるようになった。
- 新しいステートメント"action" を追加した。これは、データノードに関連付けられた操作を定義するために使用される。
- 通知をデータノードに関連付けることを許可する。
- 新しいデータ定義ステートメント"anydata"(セクション 7.10 参照)を追加した。これは、データを YANG でモデル化できる場合に"anyxml"の代わりに使用することを推奨する。
- union でデータ型の"empty" と"leafref" を許可する。
- キーに"empty" を入力できるようにする。
- 識別子を"xml" という文字列で始めることができないという制限を削除した。

NETCONF マッピングには、次の変更が加えられている:

- サーバは、YANG 1.1 モジュールのサポートを、<hello>メッセージの能力としてリストするのではなく、ietf-yang-library [RFC7895]を使用してアドバタイズする。

2. キーワード

この文書のキーワード"MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"NOT RECOMMENDED"、"MAY"、"OPTIONAL"は、BCP 14[RFC2119]に記述されているように解釈されるべきものである。

3. 用語

本書では、以下の用語を使用している:

- アクション: データツリー内のノードに定義されている操作。
- anydata: anyxml を除き、YANG でモデル化できる未知のノードセットを含むことができるデータノード。
- anyxml: 不明な XML データのチャンクを含むことができるデータノード。
- augment: 以前に定義したスキーマノードに新しいスキーマノードを追加するためのステートメント。
- 基底データ型: 派生データ型の派生元となっているデータ型。予め組み込まれたデータ型または別の派生データ型のいずれかである。
- 組み込みデータ型: uint32 や string など、YANG 言語であらかじめ定義されている YANG データ型。
- choice: 指定された複数の選択肢のうちの 1 つのみが有効なスキーマノード。
- クライアント: ネットワーク管理プロトコルを介して、サーバ上の YANG 定義データにアクセスできるエンティティ。
- 適合性: サーバがデータモデルをどれくらい正確にたどるかを表す尺度。
- container: データツリー内の最大 1 つのインスタンスに存在する内部データノード。container には値はなく、子ノードのセットがある。
- 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ステートメント。"container"、"leaf"、"leaf-list"、"list"、"choice"、"case"、"augment"、"uses"、"anydata"、"anyxml" のいずれか。
- データモデル: データモデルは、データの表現方法とアクセス方法を記述する。
- データノード: データツリーにインスタンス化できるスキーマツリー内のノード。container、leaf、leaf-list、list、anydata、anyxml のいずれか。
- データツリー: YANG でモデル化されたデータをインスタンス化したツリー。たとえば、設定データ、状態データ、設定データと状態データの組み合わせ、RPC またはアクション入力、RPC またはアクション出力、通知などである。
- 派生データ型: 組み込みデータ型(uint32 など) または別の派生データ型から派生したデータ型。
- extension: 拡張は、ステートメントに YANG 以外のセマンティクスを付与する。"extension"ステートメントは、これらのセマンティクスを表現する新しいステートメントを定義する。
- feature: モデルの一部をオプションとしてマークするメカニズム。定義は feature 名でタグ付けでき、その feature をサポートするサーバでのみ有効である。
- grouping: 再利用可能なスキーマノードのセット。これは、module 内でローカルに使用することも、そこからインポートする他の module によって使用することもできる。"grouping"ステートメントはデータ定義ステートメントではなく、スキーマツリー内のノードを定義しない。
- identifier: さまざまな種類の YANG 項目を名前で識別するために使用される文字列。
- identity: グローバルに一意で、抽象的な、型なしの名前。
- インスタンス識別子:データツリー内の特定のノードを識別するためのメカニズム。

- 内部ノード: leaf ノードではない階層内のノード。
- leaf: leaf ノードと呼ばれる、データツリー内の最大 1 つのインスタンスに存在するデータノード。leaf ノードには値はあるが、子ノードはない。
- leaf-list: leaf ノードと同様であるが、単一ノードではなく一意に識別可能なノードのセットを定義する。各ノードには値はあるが、子ノードはない。
- list: データツリー内の複数のインスタンスに存在する内部データノード。list には値はなく、子ノードのセットがある。
- 必須ノード: 必須ノードは次のいずれかである:
  * 値が"true"の"mandatory"ステートメントを持つ leaf、choce、anydata、または anyxml ノード。
  * 値がゼロより大きい"min-elements"ステートメントを持つ list ノードまたは leaf-list ノード。
  * "presence"ステートメントがなく、少なくとも 1 つの必須ノードを子として持つ container ノード。
- module: YANG module はスキーマノードの階層を定義する。その定義と定義を使用して、他の場所からインポートまたはインクルードすると、module は自己完結型で"コンパイル可能"になる。
- non-presence container: 子ノードのみを含むために存在する、独自の意味を持たない container。
- presence container: container 自体の存在が何らかの意味を持つ container。
- RPC: リモート・プロシージャ・コール。
- RPC オペレーション: 特定の操作を行うリモート・プロシージャ・コール。
- スキーマノード: スキーマツリー内のノード。action、container、leaf、leaf-list、list、choice、case、rpc、input、output、notification、anydata、anyxml のいずれか。
- スキーマノード ID: スキーマツリー内の特定のノードを識別するためのメカニズム。
- スキーマツリー: module 内で指定された定義階層。
- サーバ: 何らかのネットワーク管理プロトコルを介してクライアントに YANG 定義データへのアクセスを提供するエンティティ。
- サーバの偏差: サーバがモジュールを忠実に実装できないこと。
- submodule: 派生データ型、グループ化、データノード、RPC、アクション、および通知をモジュールに提供する部分的なモジュール定義。YANG モジュールは、複数のサブモジュールから構成できる。
- 最上位データノード: データノードと"module"または"submodule"ステートメントの間に他のデータノードがないデータノード。
- uses: "uses" ステートメントは、"grouping"ステートメントで定義されたスキーマノードのセットをインスタンス化するために使用される。インスタンシエートされたノードは、特定のニーズに合わせて調整するために、改良および拡張することができる。
- value space: データ型の場合は、データ型で許可される値のセット。leaf インスタンスまたは leaf-list インスタンスの場合はそのデータ型の値スペース。

以下の用語は[RFC6241]で定義されている:

- 設定データ
- 構成データストア
- データストア
- 状態データ

YANG を使用してモデル化すると、データストアはインスタンス化されたデータツリーとして実現される。

YANG を使用してモデル化すると、構成データストアは、構成データを含むインスタンス化されたデータツリーとして実現される。

3.1. 例に関する注記

このドキュメント全体を通して、YANG ステートメントの例が多数ある。これらの例は、特定の機能を説明するものであり、完全で有効な YANG モジュールであるとは想定されていない。

4. YANG の概要

この非規範的なセクションは、初めて読む人に YANG の概要を示すことを意図している。

4.1. 機能概要

YANG は、当初、NETCONF プロトコルのデータをモデル化するために設計された言語である。YANG モジュールは、設定、状態データ、RPC、通知など、NETCONF ベースの操作に使用できるデータの階層を定義する。これにより、NETCONF クライアントとサーバ間で送信されるすべてのデータの完全な説明が可能になる。この仕様の有効範囲外であるが、YANG は NETCONF 以外のプロトコルでも使用できる。

YANG は、データの階層構造をツリーとしてモデル化する。ツリーでは、各ノードに名前が付けられ、値または子ノードのセットのいずれかが含まれる。YANG は、ノードの明確で簡潔な説明と、それらのノード間の相互作用を提供する。

YANG は、データモデルをモジュールおよびサブモジュールに構造化する。モジュールは、他の外部モジュールから定義をインポートでき、サブモジュールからの定義を含めることができる。階層を拡張して、あるモジュールが別のモジュールで定義された階層にデータノードを追加できるようにすることができる。この拡張は条件付きで、特定の条件が満たされた場合にのみ新しいノードが表示される。

YANG データモデルは、階層内の他のノードの存在または値に基づいてノードの存在または値を制限し、データに適用される制約を記述できる。これらの制約は、クライアントまたはサーバのいずれかによって適用される。

YANG は、組み込みデータ型のセットを定義し、追加のデータ型を定義できるデータ型のメカニズムを備えている。派生データ型では、クライアントまたはサーバが適用できる範囲制限やパターン制限などのメカニズムを使用して、基本データ型の有効な値のセットを制限できる。また、ホスト名を含む文字列ベースのデータ型など、派生データ型を使用するための使用規則を定義することもできる。

YANG は、ノードの再利用可能なグループの定義を許可する。これらのグループ化を使用すると、ノードを絞り込んだり、拡張したりして、特定のニーズに合わせてノードを調整できる。派生データ型および派生グループは、1 つのモジュールで定義し、それをインポートする同じモジュールまたは別のモジュールで使用することができる。

YANG データ階層構造には、リストエントリが互いに区別するキーによって識別されるリストの定義が含まれる。このようなリストは、ユーザによってソートされるか、システムによって自動的にソートされるかのいずれかとして定義される。ユーザソート・リストの場合、リスト項目の順序を操作するための操作が定義される。

YANG モジュールは、YANG Independent Notation (YIN) (セクション 13) と呼ばれる同等の XML 構文に変換できる。これにより、XML パーサおよび Extensible Stylesheet Language Transformations (XSLT) スクリプトを使用するアプリケーションがモデル上で動作できるようになる。YANG から YIN への変換は意味的にロスレスであるため、YIN の内容を YANG に戻すことができる。

YANG は拡張可能な言語であり、標準化団体、ベンダー、個人が拡張を定義できるようにする。ステートメント構文を使用すると、これらの拡張を自然な方法で標準の YANG ステートメントと共存させることができる。一方、YANG モジュールの拡張は、読み手がそれらを認識できるように十分に目立つようになる。

YANG は、起こりうるすべての問題を解決するという傾向には反対し、問題空間を制限して、任意の XML ドキュメントや任意のデータモデルではなく、NETCONF などのネットワーク管理プロトコルのためのデータモデルの表現を可能にする。

可能な限り、YANG は Simple Network Management Protocol(SNMP)の SMIv2(Structure of Management Information version 2 [RFC2578] [RFC2579])との互換性を維持する。SMIv2 ベースの MIB モジュールは、読み取り専用のアクセス用 YANG モジュールに自動で変換できる [RFC6643]。ただし、YANG は YANG から SMIv2 への逆変換とは無関係である。

4.2. 言語の概要

このセクションでは、YANG で使用されるいくつかの重要な構文を紹介する。これは、後のセクションで言語の仕様を理解するのに役立つ。

4.2.1. モジュールとサブモジュール

YANG データモデルはモジュールで定義される。モジュールには、関連する定義のコレクションが含まれている。

モジュールには、モジュールヘッダステートメント、リビジョンステートメント、定義ステートメントの 3 種類のステートメントがある。モジュールヘッダステートメントは、モジュールを記述し、モジュール自体に関する情報を提供する。リビジョンステートメントは、モジュールの履歴に関する情報を提供する。定義ステートメントは、データモデルが定義されているモジュールの本体である。

サーバは、複数のモジュールを実装して、同じデータの複数のビューまたはサーバのデータの個別のサブセクションの複数のビューを可能にすることができる。あるいは、サーバは、使用可能なすべてのデータを定義する 1 つのモジュールのみを実装することもできる。

モジュールは、モジュール設計者のニーズに基づいて、その定義の一部をサブモジュールに分割することができる。外部ビューは、サブモジュールの有無やサイズに関係なく、1 つのモジュールの外部ビューのままである。

"import"ステートメントは、モジュールやサブモジュールが他のモジュールで定義された定義参照できるようにする。

"include"ステートメントは、モジュール内で、それに属する各サブモジュールを識別するために使用される。

4.2.2. データモデリングの基本

YANG では、データモデリング用に 4 つの主なデータノードを定義している。以下の各サブセクションでは、YANG 構文と対応する XML エンコーディングの例を示す。YANG ステートメントの構文はセクション 6.3 で定義されている。

4.2.2.1. leaf ノード

leaf インスタンスには、整数や文字列などの単純なデータが含まれる。これには、特定のタイプの値が 1 つだけあり、子ノードはない。

YANG の例:

```
leaf host-name {
  type string;
  description
    "Hostname for this system.";
}
```

XML エンコードの例:

```
<host-name>my.example.com</host-name>
```

"leaf"ステートメントはセクション 7.6 で説明する。

4.2.2.2. leaf-list ノード

leaf-list は、特定のタイプの値のシーケンスを定義する。

YANG の例:

```
leaf-list domain-search {
  type string;
  description
    "List of domain names to search.";
}
```

XML エンコードの例:

```
<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>everywhere.example.com</domain-search>
```

"leaf-list"ステートメントはセクション 7.7 で説明する。

4.2.2.3. container ノード

container は、サブツリー内の関連ノードをグループ化するために使用される。container には子ノードのみがあり、値はない。container には、任意のタイプ(leaf、list、container、leaf-list、action、および notification)の任意の数の子ノードを含めることができる。

YANG の例:

```
container system {
  container login {
    leaf message {
      type string;
      description
        "Message given at start of login session.";
     }
   }
}
```

XML エンコードの例:

```
<system>
  <login>
    <message>Good morning</message>
  </login>
</system>
```

"container"ステートメントについては、セクション 7.5 で説明する。

4.2.2.4. list ノード

list は、リストエントリのシーケンスを定義する。各エントリは container に似ており、キーリーフが定義されていれば、キーリーフの値によって一意に識別される。list は複数のキーリーフを定義でき、任意のタイプ(leaf、list、container など) の任意の数の子ノードを含むことができる。

YANG の例:

```
list user {
  key "name";
  leaf name {
    type string;
  }
  leaf full-name {
    type string;
  }
  leaf class {
    type string;
  }
}
```

XML エンコードの例:

```
<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>
```

"list"ステートメントは、セクション 7.8 で説明する。

4.2.2.5. 例モジュール

これらのステートメントは、モジュールを定義するために結合される:

```
// Contents of "example-system.yang"
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.";
    }
    leaf-list domain-search {
      type string;
      description
        "List of domain names to search.";
    }
    container login {
      leaf message {
        type string;
        description
          "Message given at start of login session.";
      }
      list user {
        key "name";
        leaf name}
          type string;
        }
        leaf full-name {
          type string;
        }
        leaf class {
          type string;
        }
      }
    }
  }
```

```
}
```

4.2.3. コンフィグレーションおよび状態データ(config)

YANG は、"config" ステートメントに基づいて、状態データと設定データをモデル化できる。ノードに"config false" というタグが付けられるとサブ階層に状態データとしてフラグが設定される。"config true" というタグが付けられるとサブ階層に設定データとしてフラグが設定される。親 container、list、およびキーリーフも報告され、状態データのコンテキストが示される。

この例では、設定された速度と観測された速度の 2 つの leaf がインターフェイスごとに定義されている。

```
list interface {
  key "name";
  config true;
  leaf name {
    type string;
  }
  leaf speed {
    type enumeration {
      enum 10m;
      enum 100m;
      enum auto;
    }
  }
  leaf observed-speed {
    type uint32;
    config false;
  }
}
```

"config"ステートメントについては、セクション 7.21.1 で説明する。

4.2.4. 組み込み型

YANG には、多くのプログラミング言語に似た一連の組み込み型があるが、ネットワーク管理の特別な要件によりいくつかの違いがある。次の表に、セクション 9 で説明する組み込み型の概要を示す:

名称	説明
binary	バイナリデータ
bits	ビットまたはフラグのセット
boolean	"true"または"false"
decimal64	64 ビット符号付き 10 進数

empty	値のない leaf
enumeration	列挙型
identityref	他のアイデンティティへの参照
instance-identifier	データツリーノードへの参照
int8	8 ビット符号付き整数
int16	16 ビット符号付き整数
int32	32 ビット符号付き整数
int64	64 ビット符号付き整数
leafref	leaf インスタンスへの参照
String	文字列
uint8	8 ビット符号なし整数
uint16	16 ビット符号なし整数
uint32	32 ビット符号なし整数
uint64	64 ビット符号なし整数
union	共用体

"type"ステートメントについてはセクション 7.4 で説明する。

4.2.5. 派生型(typedef)

YANG では、"typedef" ステートメントを使用して基底型から派生型を定義できる。基底型は、組み込み型または派生型のいずれかで、派生型の階層を許可する。派生型は、"type" ステートメントの引数として使用できる。

YANG の例:

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

XML エンコードの例:

```
<completed>20</completed>
```

"typedef"ステートメントについてはセクション 7.3 で説明する。

4.2.6. 再利用可能なノードグループ(grouping)

ノードのグループは、"grouping" ステートメントを使用して再利用可能なコレクションにアセンブルできる。グループ化は、"uses" ステートメントでインスタンス化されるノードのセットを定義する。

YANG の例:

```
grouping target {
  leaf address {
    type inet:ip-address;
    description "Target IP address.";
  }
  leaf port {
    type inet:port-number;
    description "Target port number.";
  }
}
container peer {
  container destination {
    uses target;
  }
}
```

XML エンコードの例:

```
<peer>
  <destination>
    <address>2001:db8::2</address>
    <port>830</port>
  </destination>
</peer>
```

グループ化は、使用時に再定義でき、特定のステートメントをオーバーライドできるようにすることができる。この例では、description が再定義されている:

```
container connection {
  container source {
    uses target {
      refine "address" {
```

```
        description "Source IP address.";
      }
      refine "port" {
        description "Source port number.";
      }
    }
  }
  container destination {
    uses target {
      refine "address" {
        description "Destination IP address.";
      }
      refine "port" {
        description "Destination port number.";
      }
    }
  }
}
```

"grouping"ステートメントについてはセクション 7.12 で説明する。

4.2.7. 選択肢(choice)

YANG では、互換性のないノードを"choice" ステートメントと"case" ステートメントを使用して個別の choice に分離することができる。"choice" ステートメントには、一緒に表示できないスキーマノードのセットを定義する"case" ステートメントのセットが含まれている。各"case" には複数のノードを含めることができるが、"choice" の下の 1 つの"case" にのみ各ノードを含めることができる。

choice ノードと case ノードは、スキーマツリーにのみ表示され、データツリーには表示されない。追加の階層レベルは、概念スキーマ以外には必要ない。case の存在は、その中に 1 つ以上のノードが存在することによって示される。

choice 内の case の 1 つだけがいつでも有効になる可能性があるため、1 つの case のノードがデータツリーに作成されると、他のすべての case のすべてのノードが暗黙的に削除される。サーバは制約の適用を処理し、構成内に非互換性が存在しないようにする。

YANG の例:

```
container food {
  choice snack {
    case sports-arena {
      leaf pretzel {
        type empty;
      }
      leaf beer {
        type empty;
      }
```

```
    }
    case late-night {
      leaf chocolate {
        type enumeration {
          enum dark;
          enum milk;
          enum first-available;
        }
      }
    }
  }
}
```

XML エンコードの例:

```
<food>
  <pretzel/>
  <beer/>
</food>
```

"choice"ステートメントについては、セクション 7.9 で説明する。

4.2.8. データモデルの拡張(augment)

YANG を使用すると、モジュールは、現在のモジュール(およびサブモジュール) と外部モジュールの両方を含む追加のノードをデータモデルに挿入できる。これは、ベンダーが相互運用可能な方法でベンダー固有のパラメータを標準データモデルに追加する場合などに役立つ。

"augment" ステートメントは、新しいノードが挿入されるデータモデル階層内の位置を定義し、"when" ステートメントは新しいノードが有効なときの条件を定義する。

サーバが"augment" ステートメントを含むモジュールを実装するとき、これは、拡張モジュールのサーバの実装が追加のノードを含むことを意味する。

YANG の例:

```
augment /system/login/user {
  when "class != 'wheel'";
  leaf uid {
    type uint16 {
      range "1000 .. 30000";
    }
  }
}
```

この例では、ユーザの"class" が"wheel" でない場合にのみ有効な"uid" ノードを定義する。

モジュールが別のモジュールを拡張する場合、エンコーディングに追加される XML エレメントは拡張モジュールの名前空間にある。例えば、上記の拡張が接頭辞"other" のモジュールにある場合、XML は以下のようになる:

XML エンコードの例:

```
<user>
  <name>alicew</name>
  <full-name>Alice N. Wonderland</full-name>
  <class>drop-out</class>
  <other:uid>1024</other:uid>
</user>
```

"augment"ステートメントは、セクション 7.17 で説明する。

4.2.9. オペレーション定義(rpc、action)

YANG では、オペレーションを定義できる。オペレーションの名前、入力パラメータ、および出力パラメータは、YANG データ定義ステートメントを使用してモデル化される。モジュール内の最上位レベルでのオペレーションは、"rpc" ステートメントで定義される。オペレーションは、container またはリストデータノードに関連付けることもできる。このようなオペレーションは、"action" ステートメントで定義される。

YANG トップレベルでのオペレーションの例:

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

NETCONF XML の例:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <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:netconf:base:1.0">
  <status xmlns="http://example.com/system">
    The image example-fw-2.3 is being installed.
  </status>
</rpc-reply>
```

YANG リストデータノードに関連付けられたオペレーションの例:

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

NETCONF XML の例:

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <interface xmlns="http://example.com/system">
      <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:netconf:base:1.0"
            xmlns:sys="http://example.com/system">
    <sys:packet-loss>60</sys:packet-loss>
  </rpc-reply>
```

"rpc"ステートメントはセクション 7.14 で、"action"ステートメントはセクション 7.15 で説明する。

4.2.10. 通知の定義(notification)

YANG 通知の定義を許可する。YANG データ定義ステートメントは、通知の内容をモデル化するために使用される。

YANG の例:

```
notification link-failure {
  description
    "A link failure has been detected.";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

NETCONF XML の例:

```
<notification
    xmlns="urn:ietf:params:netconf:capability:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="urn:example:system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

"notification"ステートメントは、セクション 7.16 で説明する。

5. 言語の概念

5.1. モジュールとサブモジュール

モジュールは、YANG の定義の基本単位である。モジュールは、単一のデータ・モデルを定義する。モジュールは、既存のデータ・モデルを追加のノードで拡張することもできる。

サブモジュールは、モジュールの定義を提供する部分モジュールである。モジュールには、任意の数のサブモジュールを含めることができるが、各サブモジュールは 1 つのモジュールにのみ属することができる。

YANG モジュールおよびサブモジュールの開発者は、例えば、エンタープライズ名または組織名をモジュール名のプレフィックスとして使用することによって、標準または他のエンタープライズ・モジュールと衝突する可能性が低いモジュールの名前を選択することを推奨する[RECOMMENDED]。サーバ内では、すべてのモジュール名は一意でなければならない(MUST)。

モジュールは"include" ステートメントを使用して、すべてのサブモジュールをリストする。そのモジュールに属するモジュールまたはサブモジュールは、そのモジュール内の定義およびそのモジュールに含まれるすべてのサブモジュールを参照できる。

モジュールまたはサブモジュールは、"import" ステートメントを使用して外部モジュールを参照する。モジュールまたはサブモジュール内のステートメントは、"import" ステートメントで指定されたプレフィックスを使用して外部モジュール内の定義を参照できる。

YANG バージョン 1 との下位互換性のために、サブモジュールは、そのモジュール内の他のサブモジュールを参照するために"include"ステートメントを使用してもよい[MAY]が、これは YANG バージョン 1.1 では必要ない。サブモジュールは、それが属するモジュール内、およびモジュールに含まれるすべてのサブモジュール内の任意の定義を参照できる。サブモジュールには、そのモジュールに含まれているリビジョンとは異なる他のサブモジュールのリビジョンを含めてはならない(MUST NOT)。

モジュールまたはサブモジュールに他のモジュールのサブモジュールを含めてはならず(MUST NOT)、サブモジュールは独自のモジュールをインポートしてはならない(MUST NOT)。

"import" ステートメントと"include" ステートメントは、他のモジュールから定義を利用できるようにするために使用される:

- モジュールまたはサブモジュールが外部モジュールの定義を参照するには、外部モジュールをインポートしなければならない(MUST)。
- モジュールは、そのすべてのサブモジュールを含まなければならない(MUST)。
- そのモジュールに属するモジュールまたはサブモジュールは、そのモジュール内の定義およびそのモジュールによって含まれるすべてのサブモジュールを参照してもよい[MAY]。

インポートの循環チェーンがあってはならない(MUST NOT)。例えば、モジュール"a" がモジュール"b" をインポートした場合、"b" は"a" をインポートできない。

外部モジュール内の定義が参照される時、ローカルに定義されたプレフィックスを使用し、コロン(":")と外部識別子が続かなければならない(MUST)。ローカル・モジュール内の定義への参照は、プレフィックス表記を使用してもよい[MAY]。組み込みデータ・タイプはどのモジュールにも属さず、プレフィックスもないため、組み込みデータ・タイプ(例えば int32) への参照ではプレフィックス表記を使用できない。定義への参照のための構文は、セクション 14 の規則"identifier-ref"によって正式に定義される。

5.1.1. リビジョン別インポートおよびインクルード

公開されたモジュールは、時間の経過とともに個別に進化する。この進化を可能にするために、特定のリビジョンを使用してモジュールをインポートすることができる。最初に、モジュールは、そのモジュールが書き込まれたときに現在の他のモジュールのリビジョンをインポートする。インポートされたモジュールの今後のリビジョンが公開されると、インポートするモジュールは影響を受けず、その内容は変更されない。モジュールの作成者が、インポートされたモジュールの最新のリビジョンに移動する準備ができ

たら、モジュールは更新された"import" ステートメントで再公開される。新しいリビジョンで再公開することにより、開発者はインポートされたモジュールの変更を受諾したことを明示的に示す。

サブモジュールの場合、問題は関連しているが、より単純である。サブモジュールを含むモジュールまたはサブモジュールは、含まれるサブモジュールのリビジョンを指定できる。サブモジュールが変更された場合、リビジョンごとにそれを含むモジュールまたはサブモジュールは、新しいリビジョンを参照するように更新する必要がある。

たとえば、モジュール"b" はモジュール"a" をインポートする。

```
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;
  }
}
```

"a"の開発者が新しいリビジョンを発行するとき、変更は"b"の開発者には受け入れられないかもしれない。もし新しいリビジョンが受け入れられるなら、"b"の開発者は"import"ステートメントの更新されたリビジョンで再発行できる。

モジュールが特定のリビジョンでインポートされていない場合、どのリビジョンが使用されるかは定義されていない。

5.1.2. モジュール階層

YANG では、データに複数の最上位ノードを存在する可能性ある複数の階層のデータをモデリングできる。モジュール内の各最上位データ・ノードは、個別の階層を定義する。複数の最上位ノードを持つモデルは、便利な場合があり、YANG でサポートされている。

5.1.2.1. NETCONF XML エンコーディング

NETCONF は、<config>要素と<data>要素のペイロードとして、任意の XML コンテンツを伝送できる。YANG モジュールの最上位ノードは、これらの要素内の子要素として、任意の順序でエンコードされる。このカプセル化により、対応する NETCONF メッセージが常に整形式の XML ドキュメントであることが保証される。

例えば、以下のインスタンスである:

```
module example-config {
  yang-version 1.1;
  namespace "urn:example:config";
  prefix "co";
  container system { ... }
  container routing { ... }
}
```

NETCONF で次のように符号化できる:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <!-- system data here -->
      </system>
      <routing xmlns="urn:example:config">
        <!-- routing data here -->
      </routing>
    </config>
  </edit-config>
</rpc>
```

5.2. ファイルレイアウト

YANG モジュールおよびサブモジュールは通常、ファイルに格納され、ファイルごとに"モジュール"または"サブモジュール"ステートメントがある。ファイル名は以下の形式であるべきである(SHOULD):

```
module-or-submodule-name ['@' revision-date] ( '.yang' / '.yin' )
```

module-or-submodule-name はモジュールまたはサブモジュールの名前で、オプションの"revision-date" は"revision" ステートメント(セクション 7.1.9) で定義されているモジュールまたはサブモジュールの最新のリビジョンである。

ファイル拡張子".yang"はファイルの内容が YANG 構文(セクション 6)で記述されていることを示し、".yin"はファイルの内容が YIN 構文(セクション 13)で記述されていることを示す。

YANG パーサは、この規則を使用して、インポートされたモジュールとインクルードされたサブモジュールを検索できる。

5.3. XML 名前空間

すべての YANG 定義は、モジュール内で指定される。各モジュールは、グローバルにユニークな URI [RFC3986]である個別の XML 名前空間[XML-NAMES]にバインドされる。NETCONF クライアントまたはサーバは、データの XML エンコード時に名前空間を使用する。

RFC ストリーム[RFC4844]で公開されたモジュールの XML 名前空間は、IANA によって割り当てられなければならない(MUST)。[RFC6020]のセクション 14 を参照。

プライベートモジュールの XML 名前空間は、中央レジストリーを持たないモジュールを所有する組織によって割り当てられる。名前空間 URI は、標準またはその他のエンタープライズ名前空間と衝突しないように (たとえば、名前空間のエンタープライズ名または組織名を使用して)、選択しなければならない(MUST)。"名前空間"の宣言はセクション 7.1.3 で扱う。

5.3.1. YANG XML 名前空間

YANG は、NETCONF<edit-config> operations、<error-info> content、および<action>要素の XML 名前空間を定義する。この名前空間の名前は"urn:ietf:params:xml:ns:yang:1" である。

5.4. グループ化、タイプ、および ID 名の解決

グループ化、タイプ、および ID 名は、それらが使用されるコンテキストではなく、それらが定義されているコンテキストで解決される。グループ、typedef、および ID のユーザは、元の定義によって行われたすべての参照を満たすために、モジュールをインポートしたり、サブモジュールを組み込んだりする必要はない。これは、従来のプログラミング言語での静的スコープのように動作する。

例えば、あるモジュールが、タイプが参照されるグループ化を定義している場合、グループ化が 2 番目のモジュールで使用されると、そのタイプは 2 番目のモジュールではなく、元のモジュールのコンテキストで解決される。両方のモジュールでタイプが定義されていれば、あいまいさはない。

5.5. ネストされた Typedefs およびグループ化

Typedef とグループ化は、多くの YANG ステートメントの下にネストされて表示される場合があり、それらが表示されるステートメント階層によって静的スコープを設定できる。これにより、タイプとグループを階層の最上位に配置するのではなく、それらが使用されている場所の近くで定義することができる。近接性が高いほど、読みやすくなる。

スコーピングを使用すると、異なるサブモジュール内のタイプ間で名前の競合を考慮せずにタイプを定義することもできる。タイプ名は、大きなモジュール内で名前が衝突しないように設計された先頭の文字列を追加せずに指定できる。

最後に、スコーピングにより、モジュール作成者は、タイプとグループ化をモジュールまたはサブモジュールに対して非公開にして、それらを再利用できないようにすることができる。最上位のタイプとグループ化(すなわち、"module"や"submodule"ステートメントのサブステートメントとして表示されるもの) だけがモジュールやサブモジュールの外側で使用できるので、開発者はモジュールのどの部分が外側に提示されるかをより詳細に制御でき、内部情報を非表示にする必要性をサポートし、外部と共有されるものとプライベートに保たれるものとの間の境界を維持する。

スコープ定義は、より高いスコープで定義をシャドウイングしてはならない(MUST NOT)。ステートメント階層の上位レベルに一致する ID を持つ定義がある場合、タイプまたはグループ化を定義することはできない。

プレフィックスのないタイプやグループ、あるいは現在のモジュールのプレフィックスを使用するタイプやグループ化への参照は、それぞれの先祖ステートメントの直接のサブステートメントの間で一致する"typedef" や"grouping" ステートメントを見つけることで解決される。

5.6. 適合性

モデルへの適合性とは、サーバがモデルにどれだけ正確に従うかを示す尺度である。一般的に、サーバはモデルを忠実に実装し、アプリケーションはモデルを実装するサーバを同じように扱えるようにする。モデルからの逸脱は、モデルの有用性を低下させ、それを使用するアプリケーションの脆弱性を増大させる可能性がある。YANG モデラには、3 つの適合メカニズムがある:

- モデルの基本動作
- モデルの一部であるオプション機能
- モデルからの逸脱

これらの各々を順番に検討する。

5.6.1. 基本動作

このモデルは、YANG ベースのクライアントとサーバ間の契約を定義する。この契約は、モデル化されたデータの背後にある構文と意味を相手が知っていることを両者が信用できるようにする。YANG の強みは、この契約の強さにある。

5.6.2. オプション機能

多くのモデルでは、モデラはモデルのセクションを条件付きにすることができる。サーバは、モデルのこれらの条件付き部分がサポートされているか、特定のサーバで有効かを制御する。

たとえば、syslog データモデルでは、ログをローカルに保存する機能を含めることを選択できるが、モデラは、これが可能なのはサーバにローカルストレージがある場合のみであることを認識する。ローカルストレージがない場合、アプリケーションはサーバにログを保存するように指示しない。

YANG は、"feature" と呼ばれる構文を使用して、この条件付きメカニズムをサポートする。フィーチャーは、モデラに、サーバによって制御される方法でモジュールの一部を条件付きにするためのメカニズムを提供する。このモデルは、すべてのサーバに存在するわけではない構造を表現することができる。これらの機能はモデル定義に含まれているため、一貫した表示が可能になり、どの機能がサポートされているかをアプリケーションが学習し、その動作をサーバに合わせることができる。

モジュールは、単純な文字列で識別される任意の数のフィーチャーを宣言することができ、それらのフィーチャーに基づいてモジュールの一部をオプションとすることができる。サーバがフィーチャーをサポートしている場合、モジュールの対応する部分はそのサーバで有効である。サーバがこのフィーチャーをサポートしていない場合、モジュールのこれらの部分は無効であり、アプリケーションはそれに応じて動作する必要がある。

フィーチャーは"feature"ステートメントを使用して定義される。フィーチャーの条件となるモジュール内の定義は、"if-feature"ステートメントによって示される。

詳細はセクション 7.20.1 に記載されている。

5.6.3. 逸脱

理想的な世界では、すべてのサーバが定義されたとおりに機種を実装する必要があり、機種からの逸脱は許されない。しかし、現実世界では、サーバは、書かれた通りに機種を実装することができない、あるいは設計されていないことが多い。YANG ベースの自動化がこれらのサーバの逸脱に対処するためには、サーバがそのような逸脱の詳細をアプリケーションに知らせるメカニズムが存在しなければならない。

例えば、BGP モジュールは、任意の数の BGP ピアを許可する可能性があるが、特定のサーバは、16 の BGP ピアのみをサポートする可能性がある。17 番目のピアを設定しているアプリケーションには、エラ

ーが表示される。エラーは、他のピアを追加できないことをアプリケーションに知らせるのに十分かもしれないが、もしアプリケーションがこの制限を事前に知っていて、ユーザが手戻りするのを防ぐことができれば、はるかに良いだろう。

サーバからの逸脱は、"deviation"ステートメントを使用して宣言される。このステートメントは、その引数として、スキーマツリー内のノードを識別する文字列を取る。ステートメントの内容は、モジュールで定義されているように、サーバ実装が契約から逸脱する方法を詳述する。詳細はセクション 7.20.3 に記載されている。

5.6.4. NETCONF における適合性情報の発表

この文書は、適合性情報を発表するために、以下のメカニズムを定義する。他のメカニズムは、将来の仕様によって定義される可能性がある。

NETCONF サーバは、[RFC7895]で定義されている YANG モジュール"ietf-yang-library"を実装し、実装されているすべてのモジュールを"/modules-state/module"リストに列挙することによって、それが実装しているモジュール(セクション 5.6.5 参照)を発表しなければならない(MUST)。

また、サーバは<hello>メッセージで次の機能を通知しなければならない(MUST)(改行と空白はフォーマットのためだけに使われる):

```
urn:ietf:params:netconf:capability:yang-library:1.0?
  revision=<date>&module-set-id=<id>
```

パラメータ"revision" は、サーバによって実装された"ietf-yang-library" モジュールのリビジョン日付と同じ値を持つ。このパラメータは存在しなければならない(MUST)。

パラ―メータ"module-set-id" は"ietf-yang-library" のリーフ"/modules-state/module-set-id" と同じ値を持つ。このパラメータは存在しなければならない(MUST)。

このメカニズムを使用すると、クライアントはサーバでサポートされているモジュールをキャッシュし、<hello>メッセージの"module-set-id" 値が変更された場合にのみキャッシュを更新できる。

5.6.5. モジュールの実装

モジュールのデータノード、RPC、アクション、通知、および逸脱を実装しているサーバは、モジュールを実装している。

サーバは、モジュールの複数のリビジョンを実装してはならない(MUST NOT)。

サーバがモジュール B をインポートするモジュール A を実装し、A がサーバがサポートする"augment"または"path"ステートメントで B のノードを使用する場合、サーバは、これらのノードが定義されたモジュール B のリビジョンを実装しなければならない(MUST)。これは、モジュール B がリビジョン別にインポートされているかどうかには関係ない。

サーバがモジュール C のリビジョン日付を指定せずにモジュール C をインポートするモジュール A を実装し、サーバが C を実装しない場合（たとえば、C が一部の typedef のみを定義する場合)、サーバはモジュール C を"/ modules- "ietf-yang-library"[RFC7895]の"/modules-state/module"にリストしなければならず(MUST)、このモジュールのリーフ"conformance-type"を"import"に設定しなければならない(MUST)。

サーバが"ietf-yang-library"の"/ modules-state / module"リストにモジュール C をリストし、モジュール C のリビジョン日付を指定せずに C をインポートする他のモジュール Ms がリストされている場合、サーバはモジュール Ms にリストされている C の最新リビジョンの定義を使用しなければならない(MUST)。

これらのルールの理由は、クライアントがサーバに実装されているすべてのリーフおよびリーフリストの特定のデータモデル構造およびタイプを知る必要があるためである。

例えば、以下のモジュールでは:

```
module a {
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";
  import b {
    revision-date 2015-01-01;
  }
  import c;
  revision 2015-01-01;
  feature foo;
  augment "/b:x" {
    if-feature foo;
    leaf y {
      type b:myenum;
    }
  }
  container a {
    leaf x {
      type c:bar;
    }
  }
}

module b {
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";
  revision 2015-01-01;
  typedef myenum {
    type enumeration {
      enum zero;
    }
  }
  container x {
  }
}

module b {
  yang-version 1.1;
  namespace "urn:example:b";
```

```
  prefix "b";
  revision 2015-04-04;
  revision 2015-01-01;
  typedef myenum {
    type enumeration {
      enum zero; // added in 2015-01-01
      enum one;  // added in 2015-04-04
    }
  }
  container x {  // added in 2015-01-01
    container y; // added in 2015-04-04
  }
}

module c {
  yang-version 1.1;
  namespace "urn:example:c";
  prefix "c";
  revision 2015-02-02;
  typedef bar {
    ...
  }
}

module c {
  yang-version 1.1;
  namespace "urn:example:c";
  prefix "c";
  revision 2015-03-03;
  revision 2015-02-02;
  typedef bar {
    ...
  }
}
```

モジュール"a"を実装しているが、機能"foo"をサポートしていないサーバは、モジュール"b"を実装する必要はない。

モジュール"a"のリビジョン"2015-01-01" を実装し、機能"foo" をサポートするサーバは、モジュール"b"のリビジョン"2015-01-01" または"2015-04-04" を実装できる。モジュール"b"はリビジョンによってインポートされたため、サーバが実装するモジュール"b"のリビジョンに関係なく、リーフ"/b:x/a:y"のタイプは同じである。

モジュール"a"のリビジョン"2015-01-01" を実装するサーバは、モジュール"c" のリビジョンをすべて選択し、"ietf-yang-library" の"/modules-state/module"リストにリストすること。以下の XML エンコード例は、モジュール"a"を実装するサーバの"/modules-state/module"リストの有効なデータを示す:

```
<modules-state
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
  <module-set-id>ee1ecb017370cafd</module-set-id>
  <module>
    <name>a</name>
    <revision>2015-01-01</revision>
    <namespace>urn:example:a</namespace>
    <feature>foo</feature>
    <conformance-type>implement</conformance-type>
  </module>
  <module>
    <name>b</name>
    <revision>2015-04-04</revision>
    <namespace>urn:example:b</namespace>
    <conformance-type>implement</conformance-type>
  </module>
  <module>
    <name>c</name>
    <revision>2015-02-02</revision>
    <namespace>urn:example:c</namespace>
    <conformance-type>import</conformance-type>
  </module>
</modules-state>
```

5.7. データストアの変更

データモデルを使用すると、サーバはネットワーク管理プロトコルメッセージを介して明示的に指示されない方法で構成データストアを変更できる。例えば、データモデルは、クライアントがそれを提供しない場合に、システム生成値が割り当てられるリーフを定義する場合がある。これらの変更が許される状況を指定する正式なメカニズムは、この仕様の範囲外である。

6. YANG 構文

YANG 構文は、SMIng [RFC3780]や C や C++のようなプログラミング言語と似ている。YANG はモジュールライタや YANG ツールチェーン開発者より上位のモデルの読者の時間と労力を重視しているので、この C のような構文は特にその読みやすさのために選ばれた。このセクションでは、YANG 構文について説明する。

YANG モジュールの正規文字は、Unicode および ISO/IEC 10646 [ISO.10646] 文字で、タブ、キャリッジリターン、およびラインフィードを含むが、その他の C0 制御文字、サロゲート・ブロック、および非文字は除外される。文字の構文は、セクション 14 の規則"yang-char"によって正式に定義される。

YANG モジュールとサブモジュールは、UTF-8 [RFC3629] 文字エンコーディングを使用してファイルに保存される。

YANG モジュール内の行は、キャリッジリターンとラインフィードの組み合わせで終わるか、またはラインフィードのみで終わる。キャリッジリターンが後に続かないラインフィードは、引用符で囲まれた文字列(セクション 6.1.3)の中にのみ現れる可能性がある。引用符で囲まれた文字列の中に現れるキャリッジリターンとラインフィードは、変更なしで文字列の値の一部になる。複数行の引用符で囲まれた文字列の値には、YANG モジュールの行と同じ形式の行末が含まれる。

6.1. 語彙トークン化

YANG モジュールは一連のトークンとして解析される。このセクションでは、入力ストリームからトークンを認識するためのルールについて説明する。YANG トークン化ルールは、単純かつ強力である。シンプルさは、パーサを簡単に実装できるようにする必要があることに起因している。一方、パワーは、モデラがモデルを可読形式で表現する必要があることに起因している。

6.1.1. コメント

コメントは C++ 形式である。1 行コメントは"//" で始まり、行末で終わる。ブロックコメントは"/*" で始まり、直後の"*/" で終わる。

引用符で囲まれた文字列(セクション 6.1.3)内では、これらの文字ペアはコメントの開始または終了として解釈されないことに注意する。

6.1.2. トークン

YANG のトークンは、キーワード、文字列、セミコロン(";")、または中括弧("{" または"}") のいずれかである。文字列は、引用符で囲むことも、引用符で囲まないこともできる。キーワードは、この文書で定義されている YANG キーワードのいずれか、または接頭辞 ID の後にコロン(":") が続き、その後に言語拡張キーワードが続く。キーワードは大文字と小文字が区別される。識別子の正式な定義については 6.2 節を参照。

6.1.3. 引用

引用符で囲まれていない文字列は、スペース、タブ、キャリッジリターン、またはラインフィード文字、単一引用符またはダブル引用符、セミコロン(";")、中括弧("{" または"}")、コメントシーケンス("//"、"/*"、または"*/") を含まない文字のシーケンスである。

すべてのキーワードは、引用符で囲まれていない文字列として合法的に表示される。

引用符で囲まれていない文字列内では、すべての文字が保持される。これは、バックスラッシュ文字が引用符で囲まれていない文字列では特別な意味を持たないことを意味することに注意する。

ダブルクォートされた文字列に改行が含まれ、その後に YANG ファイルのレイアウトに従ってテキストのインデントに使用されるスペースまたはタブ文字が続く場合、先頭のダブルクォート文字の列まで、または最初に出現する空白以外の最初の文字まで、この先頭の空白は文字列から取り除かれる。後続の行にあるタブ文字のうち、取り除きを調べる必要があるものは、最初に 8 つの空白文字に変換される。

ダブルクォートされた文字列に改行の前に空白またはタブ文字が含まれている場合、この末尾の空白は文字列から削除される。

単一引用符で囲まれた文字列(' ' で囲まれた文字列) は、引用符内の各文字を保持する。単一引用符で囲まれた文字列では、バックスラッシュが前に付いていても、単一引用符で囲まれた文字を使用することはできない。

ダブルクォートされた文字列("" で囲まれた) 内では、バックスラッシュ文字は特殊文字の表現を導入する。これは、バックスラッシュの直後の文字に依存する:

```
\n      newline
\t      a tab character
\"      a double quote
\\      a single backslash
```

バックスラッシュの後に他の文字を続けてはならない。

引用符で囲まれた文字列の後にプラス文字("+") が続き、その後に別の引用符で囲まれた文字列が続く場合、2 つの文字列は 1 つの文字列に連結され、複数の連結で 1 つの文字列を構築できる。引用符で囲まれた文字列とプラス文字の間には、空白、ブレイク、およびコメントを使用できる。

ダブルクォートされた文字列では、バックスラッシュでエスケープされた文字を置換する前に、空白の削除が行われる。連結は、最後の手順として実行される。6.1.3.1. 引用例

以下の文字列は同等である:

```
hello      "hello"
'hello'
"hel" + "lo"
'hel' + "lo"
```

以下の例は、いくつかの特殊な文字列を示している:

```
"\""  - string containing a double quote
'"'   - string containing a double quote
"\n"  - string containing a newline character
'\n'  - string containing a backslash followed by the character n
```

次に、不正な文字列の例を示す:

```
''''  - a single-quoted string cannot contain single quotes
"""   - a double quote must be escaped in a double-quoted string
```

以下の文字列は同等である:

```
"first line
  second line"
"first line\n" + "  second line"
```

6.2. 識別子

識別子は、異なる種類の YANG 項目を名前で識別するために使用される。各 ID は、大文字または小文字の ASCII 文字またはアンダースコア文字で始まり、その後にゼロ個以上の ASCII 文字、数字、アンダースコア文字、ハイフン、およびドットが続く。実装は 64 文字までの長さの識別子をサポートしなければならず、より長い識別子をサポートしてもよい。識別子では大文字と小文字が区別される。識別子構文は、セクション 14 の規則"識別子"によって形式的に定義される。識別子は、引用符で囲まれた文字列または引用符で囲まれていない文字列として指定できる。

6.2.1. 識別子とその名前空間

各 ID は、定義されている YANG 項目のタイプに依存する名前空間で有効である。名前空間で定義された識別子はすべてユニークでなければならない。

- すべてのモジュール名とサブモジュール名は、同じグローバル・モジュール ID 名前空間を共有する。
- モジュールおよびサブモジュールで定義されているすべての拡張名は、同じ拡張識別子名前空間を共有する。
- モジュールおよびサブモジュールで定義されているすべてのフィーチャー名は、同じフィーチャーID 名前空間を共有する。
- モジュールおよびサブモジュールで定義されているすべての ID 名は、同じ ID ID 名前空間を共有する。
- 親ノード内、またはモジュールまたはサブモジュールの最上位レベルで定義されたすべての派生タイプ名は、同じタイプ ID 名前空間を共有する。この名前空間は、親ノードまたはモジュールのすべての子孫ノードに適用される。つまり、どの子孫ノードもその typedef を使うことができ、同じ名前の typedef を定義してはならない。
- 親ノード内、またはモジュールまたはサブモジュールの最上位レベルで定義されたすべてのグループ化名は、同じグループ化 ID 名前空間を共有する。この名前空間は、親ノードまたはモジュールのすべての子孫ノードに適用される。つまり、どの子孫ノードもそのグループ化を使うことができ、同じ名前のグループ化を定義してはならない。
- 親ノード内、またはモジュールまたはサブモジュールの最上位レベルで(直接または"uses" ステートメントを使用して) 定義されたすべてのリーフ、リーフリスト、リスト、コンテナ、選択、rpcs、アクション、通知、anydatas、anyxmls は、同じ識別子名前空間を共有する。この名前空間は、親ノードがケースノードでない限り、親ノードまたはモジュールに適用される。この場合、名前空間は、ケースノードでも選択ノードでもない最も近い先祖ノードにスコープされる。
- 選択肢内のすべてのケースは、同じケース識別子名前空間を共有する。この名前空間は、親選択ノードに適用される。YANG では前方参照が許可されている。

6.3. ステートメント

YANG モジュールには、一連のステートメントが含まれている。各ステートメントは、キーワードで始まり、その後にゼロまたは 1 つの引数が続き、その後にセミコロン(";") または中括弧("{}") で囲まれたサブステートメントのブロックが続く:

```
statement = keyword [argument] (";" / "{" *statement "}")
```

引数は 6.1.2 節で定義される文字列である。

6.3.1. 言語関連

モジュールは"extension"キーワード(セクション 7.19 参照)を使用して YANG 拡張を導入することができる。拡張は"import"ステートメント(セクション 7.1.5 参照)を使用して他のモジュールからイ

ンポートすることができる。インポートされた拡張が使用される時、その拡張のキーワードは、その拡張のモジュールがインポートされたプレフィックスを使用して修飾されなければならない。拡張が定義されているモジュールで使用される場合、その拡張のキーワードは、このモジュールのプレフィックスで修飾されなければならない。

拡張の処理は、それらの拡張のサポートが、与えられた YANG パーサに対して主張されているか、それが埋め込まれているツールセットに対して主張されているかに依存する。YANG モジュールで unknown-statement (セクション 14 参照)として現れるサポートされていない拡張は、その全体が無視されてもよい。サポートされる拡張は、その拡張を規定する仕様に従って処理されなければならない。

拡張を定義するときは、その拡張を使用するモジュールが、その拡張をサポートしないアプリケーションに対しても意味を持つように注意しなければならない。

6.4. XPath の評価

YANG は、多数のノード間参照および依存関係を指定するための表記法として、XML Path Language (XPath) 1.0 [XPATH] に依存している。実装は XPath インタプリタを実装する必要はないが、データ・モデルで符号化された要件が強制されることを保証しなければならない。強制の方法は、実装上の決定である。XPath 式は構文的に正しいものでなければならず、使用されるすべてのプレフィックスは、XPath コンテキスト(セクション 6.4.1 を参照)に存在しなければならない。実装では、XPath 式を直接使用するのではなく、手動で実装することもできる。

XPath 式で使用されるデータ・モデルは、XPath 1.0 [XPATH] で使用されるものと同じで、XSLT 1.0 で使用されるものと同じルート・ノード子についての拡張子を持つ([XSLT] のセクション 3.1 を参照)。具体的には、ルート・ノードは、その子として任意の数の要素ノードを持つことができることを意味する。

データ・ツリーには、文書の順序という概念はない。実装は、何らかの文書の順序を選択する必要があるが、それがどのように行われるかは実装上の決定である。つまり、YANG モジュールの XPath 式は、特定の文書の順序に依存すべきではない。

XPath 1.0 の数値は、IEEE 754 [IEEE754-2008] 倍精度浮動小数点値である。[XPATH] のセクション 3.5 を参照する。つまり、int64、uint64、および decimal64 型の一部の値(セクション 9.2 および 9.3 を参照) は、XPath タイプで正確に表現できない。したがって、XPath 式で 64 ビットの数値を持つノードを使用する場合は、十分な注意が必要である。特に、等価性を含む数値比較は、予期しない結果をもたらす可能性がある。たとえば、次の定義を考えてみる:

```
leaf lxiv {
  type decimal64 {
    fraction-digits 18;
  }
  must ". <= 10";
}
```

10.0000000000000001 の値を持つ"lxiv"リーフのインスタンスは、正常に検証をパスする。

6.4.1. XPath コンテキスト

すべての YANG XPath 式は、以下の XPath コンテキスト定義を共有する:

- 名前空間宣言のセットは、XPath 式が指定されているモジュール内のすべての"import" ステートメントの接頭部と名前空間のペア、および"namespace" ステートメントの URI に対する "prefix" ステートメントの接頭部のセットである。
- 名前空間接頭部のない名前は、現在のノードの ID と同じ名前空間に属する。グループ内では、その名前空間は、グループ化が使用される場所の影響を受ける(セクション 7.13 を参照)。typedef 内では、その名前空間は typedef が参照される場所の影響を受ける。グループ化内で typedef が定

義され、参照されている場合、名前空間はグループ化が使用される場所の影響を受ける (セクション 7.13 参照)。

- 関数ライブラリは、[XPATH]で定義されているコア関数ライブラリであり、セクション 10 で定義されている関数である。
- 変数バインディングのセットは空である。

接頭辞のない名前を扱うメカニズムは、XPath 2.0 [XPATH2.0] から採用され、YANG の XPath 式を単純化するのに役立つ。YANG ノード ID は常に NULL 以外の名前空間 URI で修飾された名前であるため、あいまいさが発生することはない。

アクセス可能なツリーは、XPath 式を持つステートメントがどこで定義されているかによって異なる:

- XPath 式が、構成を表すデータ・ノードへのサブステートメントで定義されている場合、アクセス可能なツリーは、コンテキスト・ノードが存在するデータ・ストア内のデータである。ルート・ノードには、すべてのモジュール内のすべての最上位設定データ・ノードが子として含まれる。
- XPath 式が、状態データを表すデータ・ノードへのサブステートで定義されている場合、アクセス可能なツリーは、サーバ内のすべての状態データと、実行中の構成データ・ストアである。ルート・ノードには、すべてのモジュール内のすべての最上位データ・ノードが子として含まれる。
- XPath 式が"notification"ステートメントのサブステートメントで定義されている場合、アクセス可能なツリーは、通知インスタンス、サーバ内のすべての状態データ、および実行中の構成データ・ストアである。通知がモジュールの最上位レベルで定義されている場合、ルート・ノードには、定義されている通知を表すノードと、すべてのモジュール内のすべての最上位データ・ノードが子として含まれる。それ以外の場合、ルート・ノードには、すべてのモジュール内のすべての最上位データ・ノードが子として含まれる。
- XPath 式が"rpc" または"action" ステートメントの"input" ステートメントのサブステートで定義されている場合、アクセス可能なツリーは、RPC またはアクション操作インスタンス、サーバ内のすべての状態データ、および実行コンフィギュレーションデータストアである。ルート・ノードには、すべてのモジュールの最上位データ・ノードが子として含まれる。さらに、RPC の場合、ルート・ノードには、子として定義されている RPC 操作を表すノードもある。定義される操作を表すノードは、操作の入力パラメータを子として持つ。
- XPath 式が"rpc" または"action" ステートメントの"output" ステートメントのサブステートで定義されている場合、アクセス可能なツリーは、RPC またはアクション操作インスタンス、サーバ内のすべての状態データ、および実行コンフィギュレーションデータストアである。ルート・ノードには、すべてのモジュールの最上位データ・ノードが子として含まれる。さらに、RPC の場合、ルート・ノードには、子として定義されている RPC 操作を表すノードもある。定義されている操作を表すノードには、操作の出力パラメータが子として含まれる。

アクセス可能なツリーには、使用中のデフォルト値を持つすべてのリーフとリーフリストが存在する (セクション 7.6.1 と 7.7.2 を参照)。

アクセス可能なツリーに存在するノードが子として非プレゼンスコンテナを持つ場合、非プレゼンスコンテナもアクセス可能なツリーに存在する。

コンテキスト・ノードは YANG XPath 式によって異なり、XPath 式を持つ YANG ステートメントが定義されている場所で指定される。

6.4.1.1. 例

以下のモジュールを考えると:

```
module example-a {
  yang-version 1.1;
  namespace urn:example:a;
```

```
  prefix a;
  container a {
    list b {
      key id;
      leaf id {
        type string;
      }
      notification down {
        leaf reason {
          type string;
        }
      }
      action reset {
        input {
          leaf delay {
            type uint32;
          }
        }
        output {
          leaf result {
            type string;
          }
        }
      }
    }
  }
  notification failure {
    leaf b-ref {
      type leafref {
        path "/a/b/id";
      }
    }
  }
}
```

XML で指定された以下のデータ・ツリー:

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
```

```
    </b>
  </a>
```

/a/b[id="2"] 上の通知"down" のアクセス可能なツリーは次のとおりである:

```
  <a xmlns="urn:example:a">
    <b>
      <id>1</id>
    </b>
    <b>
      <id>2</id>
      <down>
        <reason>error</reason>
      </down>
    </b>
  </a>
  // possibly other top-level nodes here
```

/a/b[id="1"] で"reset" を実行し、"when" パラメータを"10" に設定した場合、アクセス可能なツリーは次のようになる:

```
  <a xmlns="urn:example:a">
    <b>
      <id>1</id>
      <reset>
        <delay>10</delay>
      </reset>
    </b>
    <b>
      <id>2</id>
    </b>
  </a>
  // possibly other top-level nodes here
```

このアクションのアクション出力のアクセス可能なツリーは、次のとおりである:

```
  <a xmlns="urn:example:a">
    <b>
      <id>1</id>
      <reset>
        <result>ok</result>
      </reset>
    </b>
```

```
    <b>
      <id>2</id>
    </b>
  </a>
  // possibly other top-level nodes here
```

通知"失敗"のアクセス可能なツリーは、以下のようになる:

```
  <a xmlns="urn:example:a">
    <b>
      <id>1</id>
    </b>
    <b>
      <id>2</id>
    </b>
  </a>
  <failure>
    <b-ref>2</b-ref>
  </failure>
  // possibly other top-level nodes here
```

6.5. スキーマ・ノード識別子

スキーマ・ノード識別子 は、スキーマ・ツリー内のノードを識別するストリングである。セクション 14 の"absolute-schema-nodeid"と"descendant-schema-nodeid"の規則で定義される "absolute"と"descendant"の 2 つの形式を持ち、スキーマ・ノード識別子はスラッシュ("/")で区切られた識別子のパスで構成される。絶対スキーマ・ノード識別子では、先頭のスラッシュの後の最初の識別子は、ローカル・モジュールまたはインポートされたモジュール内の任意の最上位スキーマ・ノードである。

外部モジュールで定義された識別子への参照は適切なプレフィックスで修飾されなければならない。そして、現在のモジュールとそのサブモジュールで定義された識別子への参照はプレフィックスを使用してもよい。

たとえば、最上位ノード"a" の子ノード"b" を識別するには、文字列"/a/b" を使用できる。

7. YANG ステートメント

次の項では、すべての YANG ステートメントについて説明する。

YANG でサブステートメントが定義されていないステートメントでも、ベンダー固有の拡張をサブステートメントとして持つことができる。たとえば、"description"ステートメントには YANG で定義されたサブステートメントがないが、次のステートメントは有効である。

```
description "Some text."
  ex:documentation-flag 5;
}
```

7.1. "module"ステートメント

"module"ステートメントは、モジュールの名前を定義し、モジュールに属するすべてのステートメントをグループ化する。"module"ステートメントの引数は、モジュールの名前で、その後に詳細なモジュール情報を保持するサブステートメントのブロックが続く。　モジュール名は識別子 (セクション 6.2 を参照) である。

RFC ストリーム [RFC 4844] で公開されているモジュールの名前は IANA によって割り当てられなければならない (MUST) 。[RFC 6020] のセクション 14 を参照のこと。

プライベートモジュール名は、中央レジストリなしでモジュールを所有する組織によって割り当てられる。モジュールに名前を付ける方法についてはセクション 5.1 を参照のこと。

通常、モジュールのレイアウトは次のとおりである。

```
module <module-name> {
   // header information
   <yang-version statement>
   <namespace statement>
   <prefix statement>

   // linkage statements
   <import statements>
   <include statements>

   // meta-information
   <organization statement>
   <contact statement>
   <description statement>
   <reference statement>

   // revision history
   <revision statements>

   // module definitions
   <other statements>
}
```

7.1.1. module のサブステートメント

サブステートメント	セクション	基数
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

7.1.2. "yang-version"ステートメント

"yang-version"ステートメントは、モジュールの開発に使用された YANG 言語のバージョンを指定する。ステートメントの引数は文字列である。この仕様に基づいて定義された YANG モジュールの値"1.1"を含まなければならない (MUST) 。

"yang-version"ステートメントを含まないモジュールまたはサブモジュール、または値"1"を含むモジュールまたはサブモジュールは、[RFC 6020]で定義されている。

"yang-version"ステートメントを"1.1" (ここで定義されたバージョン)以外のバージョンで処理することは、本仕様の範囲外である。上位バージョンを定義する文書は、そのような上位バージョンの下位互換性を定義する必要がある。

YANG バージョン 1 と 1.1 の互換性については、セクション 12 を参照のこと。

7.1.3. "namespace"ステートメント

"namespace"ステートメントは、モジュールによって定義されるすべての識別子が XML エンコーディングで修飾される XML 名前空間を定義する。ただし、グループ内で定義されるデータノード、アクションノード、および通知ノードの識別子は例外である(詳細はセクション 7.13 を参照)。"namespace"ステートメントの引数は、名前空間の URI である。
セクション 5.3 も参照のこと。

7.1.4. "prefix"ステートメント

"prefix"ステートメントは、モジュールおよびその名前空間に関連付けられたプレフィックスを定義するために使用される。"prefix"ステートメントの引数は、モジュールにアクセスするためのプレフィックスとして使用されるプレフィックス文字列である。プレフィックス文字列は、例えば"if:ifName"のように、モジュールに含まれる定義を参照するためにモジュールと共に使用されるかもしれない (MAY) 。プレフィックスは識別子(セクション 6.2 を参照)である。

"module"ステートメント内で使用する場合、"prefix"ステートメントは、このモジュールのインポート時に使用することが推奨されるプレフィックスを定義する。

NETCONF XML の可読性を向上させるために、プレフィックスを使用する XML または XPath を生成する NETCONF クライアントまたはサーバは、衝突がない限り、モジュールによって定義されたプレフィックスを XML 名前空間プレフィックスとして使用すべきである (SHOULD) 。

"import"ステートメント内で使用される場合、"prefix"ステートメントは、インポートされたモジュール内の定義にアクセスするときに使用されるプレフィックスを定義する。インポートされたモジュールの識別子への参照が使用される場合、インポートされたモジュールのプレフィックス文字列とそれに続くコロン (":") および識別子が、例えば、"if:ifIndex"が使用される。YANG モジュールの可読性を向上させるために、モジュールがインポートされる時、競合がない限り、モジュールによって定義されたプレフィックスが使われるべきである (SHOULD) 。衝突がある場合、すなわち、両方が同じプレフィックスを定義した 2 つの異なるモジュールがインポートされる場合、それらのうち少なくとも 1 つは異なるプレフィックスでインポートされなければならない (MUST) 。

モジュール自身のプレフィックスを含むすべてのプレフィックスは、モジュールまたはサブモジュール内で一意でなければならない (MUST)。

7.1.5. "import"ステートメント

"import"ステートメントは、あるモジュールの定義を別のモジュールまたはサブモジュール内で使用できるようにする。引数はインポートするモジュールの名前で、ステートメントの後には詳細なインポート情報を保持するサブステートメントのブロックが続く。モジュールをインポートする場合、インポートするモジュールは次のようになる。

- インポートされたモジュールまたはそのサブモジュールの最上位レベルで定義されている grouping と typedef を使用する。

- インポートされたモジュールまたはそのサブモジュールで定義されている任意の extension、feature、および identity を使用する。
- インポートしたモジュールのスキーマツリー内の任意のノードを"must"、"path"、および"when"ステートメントで使用するか、"augment"および"deviation"ステートメントでターゲットノードとして使用する。

必須の"prefix"サブステートメントは、インポートするモジュールまたはサブモジュールにスコープされるインポートされたモジュールのプレフィックスを割り当てる。異なるモジュールからインポートするために複数の"import"ステートメントを指定できる。

オプションの"revision-date"サブステートメントが存在する場合、ローカルモジュール内の定義によって参照されるすべての typedef、grouping、extension、feature、および identity は、インポートされたモジュールの指定されたリビジョンから取得される。　インポートされたモジュールの指定されたリビジョンが存在しない場合は、エラーになる。"revision-date"サブステートメントが存在しない場合、モジュールのどのリビジョンから取得するかは定義されない。

異なるプレフィックスが使用されている場合は、同じモジュールの複数のリビジョンをインポートできる。

サブステートメント	セクション	基数
description	7.21.3	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision-date	7.1.5.1	0..1

import のサブステートメント

7.1.5.1. import の"revision-date"ステートメント

import の"revision-date"ステートメントは、インポートするモジュールのバージョンを指定するために使用される。

7.1.6. "include"ステートメント

"include"ステートメントは、サブモジュールのコンテンツをそのサブモジュールの親モジュールで使用できるようにするために使用される。引数は、含めるサブモジュールの名前である識別子である。モジュールは、"belongs-to"ステートメント(セクション 7.2.2 を参照)で定義されているように、そのモジュールに属するサブモジュールのみを含めることができる。

モジュールがサブモジュールを含む場合、モジュールのノード階層にサブモジュールの内容を組み込む。

YANG バージョン 1 との下位互換性のため、サブモジュールは同じモジュールに属する別のサブモジュールを含むことができるが、YANG バージョン 1.1 (セクション 5.1 を参照)では必要ない。

オプションの"revision-date"サブステートメントが存在する場合、サブモジュールの指定されたリビジョンがモジュールに含まれる。指定されたサブモジュールのリビジョンが存在しない場合はエラーになりる。"revision-date"サブステートメントが存在しない場合、サブモジュールのどのリビジョンが含まれるかは定義されない。

同じサブモジュールの複数のリビジョンを含めてはならない (MUST NOT) 。

サブステートメント	セクション	基数

description	7.21.3	0..1
reference	7.21.4	0..1
revision-date	7.1.5.1	0..1

include のサブステートメント

7.1.7. "organization"ステートメント

"organization"ステートメントは、このモジュールの責任者を定義する。引数は、このモジュールが開発された組織のテキスト記述を指定するために使用される文字列である。

7.1.8. "contact"ステートメント

"contact"ステートメントは、モジュールの連絡先情報を提供する。引数は、名前、住所、電話番号、電子メールアドレスなど、このモジュールに関する技術的な問い合わせの送信先となる個人の連絡先情報を指定するために使用する文字列である。

7.1.9. "revision"ステートメント

"revision"ステートメントは、最初のリビジョンを含むモジュールの編集リビジョン履歴を指定する。一連の"revision"ステートメントは、モジュール定義の変更を詳細に示す。引数は"YYYY-MM-DD"形式の日付文字列で、その後に詳細なリビジョン情報を保持するサブステートメントのブロックが続く。モジュールは少なくとも一つの"revision"ステートメントを持つべきである (SHOULD) 。すべての発行された編集上の変更に対して、すべての改訂が新しい年代順になるように、改訂シーケンスの前に新しいものを追加すべきである (SHOULD) 。

7.1.9.1. revision のサブステートメント

サブステートメント	セクション	基数
description	7.21.3	0..1
reference	7.21.4	0..1

7.1.10. 使用例

次の例は、 [RFC 6991]に依存している。

```
module example-system {
  yang-version 1.1;
  namespace "urn:example:system";
  prefix "sys";
  import ietf-yang-types {
    prefix "yang";
    reference "RFC 6991: Common YANG Data Types";
  }
  include example-types;
  organization "Example Inc.";
  contact
```

```
      "Joe L. User
       Example Inc.
       42 Anywhere Drive
       Nowhere, CA 95134
       USA
       Phone: +1 800 555 0100
       Email: joe@example.com";
    description
      "The module for entities implementing the Example system.";
    revision 2007-06-09 {
      description "Initial revision.";
    }
    // definitions follow...
  }
```

7.2. "submodule"ステートメント

YANG の基本ユニットはモジュールであるが、YANG モジュール自体はいくつかのサブモジュールから構成することができる。サブモジュールを使用すると、モジュール設計者は複雑なモデルを複数の部分に分割して、すべてのサブモジュールがサブモジュールを含むモジュールによって定義される単一の名前空間に含めることができる。

"submodule"ステートメントはサブモジュールの名前を定義し、サブモジュールに属するすべてのステートメントをグループ化する。"submodule"ステートメントの引数は、サブモジュールの名前で、その後に詳細なサブモジュール情報を保持するサブステートメントのブロックが続く。サブモジュール名は識別子(セクション 6.2 項を参照)である。

RFC ストリーム[RFC 4844]で公開されているサブモジュールの名前は IANA によって割り当てられなければならない (MUST) 。[RFC 6020] のセクション 14 を参照。

プライベートサブモジュール名は、中央レジストリなしでサブモジュールを所有する組織によって割り当てられる。サブモジュールに名前を付ける方法についてはセクション 5.1 を参照のこと。

通常、サブモジュールのレイアウトは次のとおりである。

```
  submodule <module-name> {
    <yang-version statement>
    // module identification
    <belongs-to statement>
    // linkage statements
    <import statements>
    // meta-information
    <organization statement>
    <contact statement>
    <description statement>
    <reference statement>
    // revision history
    <revision statements>
    // module definitions
    <other statements>
```

```
}
```

7.2.1. submodule のサブステートメント

サブステートメント	セクション	基数
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
belongs-to	7.2.2	1
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
organization	7.1.7	0..1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

7.2.2. "belongs-to"ステートメント

"belongs-to"ステートメントは、サブモジュールが属するモジュールを指定する。引数は、モジュールの名前である識別子である。

サブモジュールは、それが属するモジュールまたはそのモジュールに属する別のサブモジュールのいずれかによってのみ含まれなければならない（MUST）。

必須"prefix"サブステートメントは、サブモジュールが属するモジュールにプレフィクスを割り当てる。サブモジュールが属するモジュール内のすべての定義およびそのすべてのサブモジュールには、プレフィクスを使用してアクセスできる。

サブステートメント	セクション	基数
prefix	7.1.4	1

belongs-to のサブステートメント

7.2.3. 使用例

```
submodule example-types {
  yang-version 1.1;
  belongs-to "example-system" {
    prefix "sys";
  }
  import ietf-yang-types {
    prefix "yang";
  }
  organization "Example Inc.";
  contact
    "Joe L. User
    Example Inc.
    42 Anywhere Drive
    Nowhere, CA 95134
    USA
    Phone: +1 800 555 0100
    Email: joe@example.com";
  description
    "This submodule defines common Example types.";
  revision "2007-06-09" {
    description "Initial revision.";
  }
  // definitions follow...
}
```

7.3. "typedef"ステートメント

"typedef"ステートメントは、セクション 5.5 の規則に従って、モジュールまたはサブモジュール、およびそこからインポートする他のモジュールでローカルに使用できる新しい型を定義する。新しい型は

"派生型"と呼ばれ、派生元の型は"基底型"と呼ばれます。派生型はすべて YANG 組み込み型にまでさかのぼることができる。

"typedef"ステートメントの引数は、定義される型の名前である識別子であり、その後に詳細な typedef 情報を保持するサブステートメントのブロックが続く必要がある (MUST) 。

型の名前は YANG 組み込み型であってはならない (MUST NOT) 。typedef が YANG モジュールまたはサブモジュールのトップレベルで定義される場合、定義される型の名前はモジュール内で一意でなければならない (MUST) 。

7.3.1. typedef のサブステートメント

サブステートメント	セクション	基数
default	7.3.4	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.3.2	1
units	7.3.3	0..1

7.3.2. typedef の"type"ステートメント

存在しなければならない"type"ステートメントは、この型が派生する基底型を定義する (MUST) 。詳細はセクション 7.4 を参照のこと。

7.3.3. "units"ステートメント

"units"ステートメントはオプションで、型に関連付けられた単位のテキスト定義を含む文字列を引数として取る。

7.3.4. typedef の"default"ステートメント

"default"ステートメントは、新しい型のデフォルト値を含む文字列を引数として取る。

"default"ステートメントの値は、"type"ステートメントで指定された型に従って有効でなければなりません (MUST) 。

基底型にデフォルト値があり、新しい派生型で新しいデフォルト値が指定されていない場合、基底型のデフォルト値は新しい派生型のデフォルト値にもなる。

派生型またはリーフ定義で指定された新しい制約に従って型のデフォルト値が有効でない場合、派生型またはリーフ定義は、制約と互換性のある新しいデフォルト値を指定しなければならない (MUST) 。

7.3.5. 使用例

```
typedef listen-ipv4-address {
  type inet:ipv4-address;
  default "0.0.0.0";
}
```

7.4. "type"ステートメント

"type"ステートメントは、引数として、YANG 組み込み型 (セクション 9 を参照) または派生型 (セクション 7.3 を参照) の名前である文字列と、その型にさらに制限を加えるために使用されるサブステートメントのオプションブロックを受け取る。

適用できる制限は、制限する型によって異なる。すべての組み込み型の制限ステートメントについては、セクション 9 のサブセクションで説明する。

7.4.1. type のサブステートメント

サブステートメント	セクション	基数
base	7.18.2	0..n
bit	9.7.4	0..n
enum	9.6.4	0..n
fraction-digits	9.3.4	0..1
length	9.4.4	0..1
path	9.9.2	0..1
pattern	9.4.5	0..n
range	9.2.4	0..1
require-instance	9.9.3	0..1
type	7.4	0..n

7.5. "container"ステートメント

"container"ステートメントは、スキーマツリー内の内部データノードを定義するために使用される。1 つの引数 (識別子) の後に、詳細なコンテナ情報を保持するサブステートメントのブロックが続く。

コンテナノードには値がないが、データツリーに子ノードのリストがある。子ノードはコンテナのサブステートメントで定義される。

7.5.1. 存在のあるコンテナ

YANG は、データノードの階層を編成するためだけに存在するコンテナと、データツリー内に存在することが明示的な意味を持つコンテナの 2 つのスタイルのコンテナをサポートしている。

最初のスタイルでは、コンテナはそれ自体の意味を持たず、子ノードを含むためだけに存在する。特に、子ノードを持たないコンテナノードが存在することは、意味的にはコンテナノードが存在しないことと同じである。YANG ではこのスタイルを"非存在コンテナ"と呼んでいる。これが既定のスタイルである。

たとえば、Synchronous Optical Network (SONET) インターフェイスのスクランブリングオプションのセットを"scrambling"コンテナ内に配置して、設定階層の編成を強化し、これらのノードをまとめておくことができる。"scrambling"ノード自体には意味がないため、ノードが空になったときにノードを削除すると、ユーザはこのタスクを実行できなくなる。

2 番目のスタイルでは、コンテナ自体の存在が何らかの意味を持ち、1 ビットのデータを表す。

構成データの場合、コンテナは構成ノブとしても、関連する構成ノードを編成する手段としても機能する。これらのコンテナは明示的に作成および削除される。

YANG はこのスタイルを"プレゼンスコンテナ"と呼び、ノードの存在の意味を示すテキスト文字列を引数として取る"presence"ステートメントを使用して示される。

たとえば、"ssh"コンテナは、Secure SHell (SSH) を使用してサーバにログインする機能を有効にすることができますが、接続レートや再試行制限などの SSH 関連の設定ノブを含めることもできる。

"presence"ステートメント(セクション 7.5.5 を参照)は、データツリー内のコンテナの存在に意味を与えるために使用される。

7.5.2. container のサブステートメント

サブステートメント	セクション	基数
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
notification	7.16	0..n
presence	7.5.5	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1

7.5.3. "must"ステートメント

"must"ステートメントは、XPath 式を含む文字列(セクション 6.4 を参照)を引数として取る。有効なデータに対する制約を正式に宣言するために使用される。この制約はセクション 8 の規則に従って強制される。

データストアが検証されると、すべての"must"制約は、アクセス可能なツリー(セクション 6.4.1 を参照)の各ノードに対して一度だけ概念的に評価される。

そのような制約はすべて、データが有効であるために"true"と評価しなければならない (MUST) 。

XPath 式は、セクション 6.4.1 の定義に加えて、次のコンテキストで概念的に評価される。

- "must"ステートメントが"notification"ステートメントのサブステートメントである場合、コンテキストノードはアクセス可能なツリー内の通知を表すノードである。
- "must"ステートメントが"input"ステートメントのサブステートメントである場合、コンテキストノードはアクセス可能なツリー内の操作を表すノードである。
- "must"ステートメントが"output"ステートメントのサブステートメントである場合、コンテキストノードはアクセス可能なツリー内の操作を表すノードである。
- それ以外の場合、コンテキストノードは"must"ステートメントが定義されているアクセス可能なツリー内のノードである。

XPath 式の結果は、標準 XPath 規則を使用してブール値に変換される。

データツリー内のすべてのリーフ値は概念的には正規形式(セクション 9.1 を参照)で格納されるため、XPath 比較は正規値に対して行われることに注意のこと。

また、XPath 式は概念的に評価されることにも注意のこと。これは、実装がサーバで XPath エバリュエーターを使用する必要がないことを意味する。評価が実際にどのように行われるかは、実装の決定である。

7.5.4. must のサブステートメント

サブステートメント	セクション	基数
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

7.5.4.1. "error-message"ステートメント

"error-message"ステートメントはオプションで、文字列を引数として取る。制約が"false"と評価された場合、文字列は NETCONF の<rpc-error>に<error-message>として渡される。

7.5.4.2. "error-app-tag"ステートメント

"error-app-tag"ステートメントはオプションで、文字列を引数として取る。制約が"false"と評価される場合、文字列は NETCONF の<rpc-error>の<error-app-tag>として渡される。

7.5.4.3. must と error-message の使用例

```
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";
      }
    }
```

7.5.5. "presence"ステートメント

"presence"ステートメントは、データツリー内のコンテナの存在に意味を割り当てる。　ノードの存在が意味する内容をテキストで記述した文字列を引数として取る。

コンテナに"presence"ステートメントがある場合、データツリー内のコンテナの存在には何らかの意味がある。それ以外の場合、コンテナはデータに何らかの構造を与えるために使用され、それ自体は意味を持たない。詳細はセクション 7.5.1 を参照のこと。

7.5.6. コンテナの子ノードステートメント

コンテナ内では、"container"、"leaf"、"list"、"leaf-list"、"uses"、"choice"、"anydata"、"anyxml"ステートメントを使用して、コンテナの子ノードを定義できる。

7.5.7. XML エンコーディング規則

コンテナノードは XML 要素としてエンコードされる。要素のローカル名はコンテナの識別子であり、その名前空間はモジュールの XML 名前空間である(セクション 7.1.3 を参照)。

コンテナの子ノードは、コンテナ要素のサブ要素としてエンコードされる。コンテナが RPC またはアクションの入力または出力パラメータを定義している場合、これらのサブ要素は"container"ステートメント内で定義されているのと同じ順序でエンコードされる。それ以外の場合、サブ要素は任意の順序でエンコードされる。

サブ要素とコンテナの間の空白は重要ではない。つまり、実装はサブ要素の間に空白文字を挿入してもよい (MAY) 。

非存在コンテナに子ノードがない場合、そのコンテナは XML エンコーディングで存在する場合と存在しない場合がある。

7.5.8. NETCONF<edit-config>操作

コンテナの XML 要素の"operation"属性([RFC6241]のセクション 7.2 を参照)を使用し<edit-config>によりコンテナの作成、削除、置換、修正ができる。

コンテナに"presence"ステートメントがなく、最後の子ノードが削除された場合、NETCONF サーバはコンテナを削除してもよい (MAY) 。

NETCONF サーバが<edit-config>要求を処理する場合、コンテナノードのプロシージャの要素は次のとおりである。

- 操作が"merge"または"replace"で、ノードが存在しない場合は作成される。
- 操作が"create"の場合、ノードが存在しなければ作成される。ノードがすでに存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、ノードが存在すれば削除される。ノードが存在しない場合は、"data-missing"エラーが返される。

7.5.9. 使用例

次のコンテナ定義を指定する。

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

対応する XML インスタンスの例:

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

<ssh>要素が存在するため、SSH は有効である。

<edit-config>を使用してコンテナを削除するには:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh nc:operation="delete"/>
        </services>
      </system>
```

```
        </config>
      </edit-config>
    </rpc>
```

7.6. "leaf"ステートメント

"leaf"ステートメントは、スキーマツリー内のリーフノードを定義するために使用される。1 つの引数 (識別子) の後に、詳細なリーフ情報を保持するサブステートメントのブロックが続く。

データツリー内において、リーフノードは値をもつが、子ノードをもたない。概念的には、データツリーの値は常に標準形式(セクション 9.1 を参照)である。

リーフノードは、データツリーの 0 または 1 つのインスタンスに存在する。

"leaf"ステートメントは、特定の組み込み型または派生型のスカラー変数を定義するために使用される。

7.6.1. leaf のデフォルト値

リーフのデフォルト値は、データツリーにリーフが存在しない場合にサーバが使用する値である。デフォルト値の使用は、非存在コンテナではないスキーマツリー内でリーフに最も近い祖先ノードに依存する(セクション 7.5.1 を参照)。

- そのような祖先がスキーマツリーに存在しない場合、デフォルト値を使用しなければなりません (MUST) 。
- そうでない場合、もしこの祖先が case ノードであれば、case からのノードがデータツリーに存在するか、case ノードが choice のデフォルトの case であり、他の case からのノードがデータツリーに存在しない場合には、デフォルト値を使用しなければならない (MUST) 。
- そうでなければ、もし祖先ノードがデータツリーに存在するなら、デフォルト値を使わなければなりません (MUST) 。

このような場合、デフォルト値は使用中と呼ばれる。

リーフまたはその先祖のいずれかに"when"条件または"if-feature"式があり、その評価結果が"false"である場合、既定値は使用されない。

デフォルト値が使用されている場合、サーバは、デフォルト値を値とするリーフがデータツリーに存在するかのように動作しなければならない (MUST) 。

リーフに"default"ステートメントがある場合、リーフのデフォルト値は"default"ステートメントの値になる。そうでない場合、リーフの型にデフォルト値があり、リーフが必須ではない場合、リーフのデフォルト値は型のデフォルト値になる。それ以外の場合、リーフはデフォルト値をもたない。

7.6.2. leaf のサブステートメント

サブステートメント	セクション	基数
config	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n

mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1

7.6.3. leaf の"type"ステートメント

"type"ステートメントは、必ず存在し、既存の組み込み型または派生型の名前を引数として取る(MUST)。オプションのサブステートメントは、この型の制限を指定する。詳細はセクション 7.4 を参照のこと。

7.6.4. leaf の"default"ステートメント

"default"ステートメントは、リーフの既定値を含む文字列を引数として取る。

"default"ステートメントの値は、リーフの"type"ステートメントで指定されている型に従って有効でなければならない(MUST)。

"default"ステートメントは、"mandatory"が"true"であるノード上に存在してはならない(MUST NOT)。

デフォルト値の定義は、"if-feature"ステートメントでマークしてはならない(MUST NOT)。たとえば、次は不正である。

```
leaf color {
  type enumeration {
    enum blue { if-feature blue; }
    ...
  }
  default blue; // illegal - enum value is conditional
}
```

7.6.5. leaf の"mandatory"ステートメント

"mandatory"ステートメントはオプションで、文字列"true",または"false"を引数として受け取り、有効なデータに制約を設定する。指定しない場合のデフォルトは"false"である。

"mandatory"が"true"の場合、制約の動作は、非存在コンテナではないスキーマツリー内でリーフに最も近い祖先ノードの型に依存する(セクション 7.5.1 を参照)。

- そのような祖先がスキーマツリーに存在しない場合、リーフは存在しなければならない(MUST)。
- そうでなければ、この祖先が case ノードである場合、その case からのノードがデータツリーに存在するならば、そのリーフが存在しなければならない(MUST)。

- そうでなければ、データツリーに祖先ノードが存在する場合、リーフが存在しなければならない(MUST)。

この制約はセクション 8 の規則に従って強制される。

7.6.6. XML エンコーディング規則

リーフノードは XML 要素としてエンコードされる。要素のローカル名はリーフの識別子であり、その名前空間はモジュールの XML 名前空間(セクション 7.1.3 を参照)である。

リーフノードの値は型に従って XML にエンコードされ、要素内の文字データとして送信される。例についてはセクション 7.6.8 を参照すること。

7.6.7. NETCONF<edit-config>操作

NETCONF サーバが<edit-config>要求を処理する場合、リーフノードのプロシージャの要素は以下のようになる。

- 操作が"merge"または"replace"の場合、ノードが存在しない場合は作成され、その値は XML RPC データで見つかった値に設定される。
- 操作が"create"の場合、ノードが存在しなければ作成される。ノードがすでに存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、ノードが存在すれば削除される。ノードが存在しない場合は、"data-missing"エラーが返される。

7.6.8. 使用例

次の"leaf"ステートメントを前に定義した"ssh"コンテナ(セクション 7.5.9 を参照)に配置する。

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

対応する XML インスタンスの例:

```
<port>2022</port>
```

<edit-config>を使用してリーフの値を設定するには

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
```

```
      <config>
        <system xmlns="urn:example:config">
          <services>
            <ssh>
              <port>2022</port>
            </ssh>
          </services>
        </system>
      </config>
    </edit-config>
  </rpc>
```

7.7. "leaf-list"ステートメント

"leaf"ステートメントを使用して特定の型の単純なスカラー変数を定義する場合、"leaf-list"ステートメントを使用して特定の型の配列を定義する。　"leaf-list"ステートメントは、識別子である引数をとり、その後に詳細な leaf-list 情報を保持するサブステートメントのブロックを続ける。設定データでは、leaf-list の値は一意でなければならない。

デフォルト値の定義は、"if-feature"ステートメントでマークしてはならない。

概念的に、データツリーの値は正規形(セクション 9.1 参照)でなければならない。

7.7.1. 順序

YANG は、list 内のエントリの順序付けと leaf-list の順序付けの 2 つのスタイルをサポートしている。　多くの list では、list・エントリの順序は list の構成の実装に影響を与えず、サーバは list・エントリを妥当な順序で自由にソートできる。list の"description"文字列は、サーバ実装者に順序を提案する。YANG はこの list 形式を"ordered-by system"と呼んでいる。そのような list は"ordered-by system"とステートメントで示される。

たとえば、有効なユーザの list は通常、アルファベット順にソートされる。これは、構成に表示されるユーザの順序が、ユーザのアカウントの作成に影"響を与えないためである。

list のもう 1 つのスタイルでは、list・エントリの順序が list の構成の実装に重要であり、ユーザがエントリの順序付けを担当し、サーバはその順序を維持する。　YANG はこの list 形式を"ユーザ順序"と呼んでいる。そのような list は"ordered-by user"というステートメントで示される。

たとえば、パケットフィルタエントリが着信トラフィックに適用される順序は、そのトラフィックのフィルタリング方法に影響する場合がある。　ユーザは、すべての TCP トラフィックを廃棄するフィルタエントリを、信頼できるインターフェイスからすべてのトラフィックを許可するフィルタエントリの前に適用するか、後に適用するかを決定する必要がある。　順序の選択は重要である。

YANG は、NETCONF の<edit-config>操作の中で、ユーザ順序 list の list エントリの順序を制御するための豊富な機能を提供する。　list エントリは、挿入または並べ替えたり、list の最初または最後のエントリとして配置したり、別の特定のエントリの前または後に配置したりできる。"ordered-by"ステートメントは、セクション 7.7.7 で説明される。

7.7.2. leaf-list のデフォルト値

leaf-list のデフォルト値は、leaf-list がデータツリーに存在しない場合にサーバが使用する値である。　デフォルト値の使用方法は、非存在コンテナではないスキーマツリー内で leaf-list の最も近い祖先ノードに依存する(セクション 7.5.1 参照)。

o　そのような祖先ノードがスキーマツリーに存在しない場合、デフォルト値を使用しなければならない。

- そうでない場合、もしこの祖先ノードが case ノードであれば、case ノードがデータツリーに存在するか、case ノードが choice のデフォルトの case であり、他の case からのノードがデータツリーに存在しない場合には、デフォルト値を使用しなければならない。
- そうでなければ、もし祖先ノードがデータツリーに存在するなら、デフォルト値を使わなければならない。　この場合、デフォルト値は使用中であると言う。

leaf-list またはその祖先ノードに"when"条件または"if-feature"式があり、その評価結果が"false"である場合、デフォルト値は使用されない。

デフォルト値が使用されている場合、サーバは、leaf-list がその値としてデフォルト値を持つデータツリーに存在するかのように動作しなければならない。

leaf-list に"default"ステートメントが含まれている場合、leaf-list のデフォルト値は"default"ステートメントの値になる。leaf-list がユーザ指定の場合、デフォルト値は"default"ステートメントの順序で使用される。　そうでない場合、leaf-list のタイプにデフォルト値があり、leaf-list に値が一つ以上の"min-elements"ステートメントがない場合、leaf-list のデフォルト値はタイプのデフォルト値のインスタンスになる。　それ以外の場合、leaf-list にはデフォルト値はない。

7.7.3. leaf-list の Substatements

substatement	section	cardinality
config	7.21.1	0..1
default	7.7.4	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.4	1
units	7.3.3	0..1
when	7.21.5	0..1

7.7.4. leaf-list の"default"ステートメ"ント

"default"ステートメントはオプションで、引数として leaf-list のデフォルト値を含むストリングを取る。

"default"ステートメントの値は、leaf-list の"type"ステートメントで指定されたタイプに従って有効でなければならない。

"default"ステートメントは、"min-elements"がひとつ以上の値を持つノード上に存在してはならない。

7.7.5. "min-elements"ステートメント

"min-elements"ステートメントはオプションで、有効な list エントリに制約を設定する負でない整数を引数として取る。　有効な leaf-list または list は、少なくとも min-elements エントリを持たなければならない。

"min-elements"ステートメントがない場合は、デフォルトの 0 になる。

制約の動作は、非存在コンテナではないスキーマツリー内で leaf-list または list の最も近い祖先ノードの型によって異なる (セクション 7.5.1 を参照)。

- スキーマツリーにそのような祖先ノードが存在しない場合、制約が適用される。
- そうでない場合、この祖先ノードが case ノードであれば、case ノードから他のノードが存在する場合に制約が適用される。
- それ以外の場合は、親ノードが存在する場合に強制される。

この制約はさらにセクション 8 の規則に従って強制される。

7.7.6. "max-elements"テートメント

"max-elements"ステートメントはオプションで、引数として正の整数または文字列"unbounded"を取り、有効な list エントリに制約を設定する。　有効な leaf-list または list には、常に max-elements までのエントリが含まれる。

"max-elements"ステートメントがない場合は、デフォルトで"unbounded"に設定される。

"max-elements"制約は、セクション 8 の規則に従って強制される。

7.7.7. "ordered-by"ステートメント

"ordered-by"ステートメントは、list 内のエントリの順序をユーザとシステムのどちらが決定するかを定義する。　引数は、文字列"system"または"user"のいずれかである。　存在しない場合、順序はデフォルトの"system"になる。

list が状態データ、RPC 出力パラメータ、または通知コンテンツを表す場合、このステートメントは無視される。　詳細はセクション 7.7.1 を参照のこと。

7.7.7.1.　ordered-by system

list 内のエントリは、システムによって決定された順序に従って並べられる。　list の"description"文字列は、サーバ実装者に順序を提案するが、そうでない場合、実装は任意の順序でエントリを順序付けることができる。　実装は、データがどのように作成されたかにかかわらず、同じデータに対して同じ順序を使用すべきである。　決定論的順序を使用すると、"diff"のような単純なツールを使用して比較が可能になる。　これがデフォルトの順序である。

7.7.7.2.　ordered-by user

list 内のエントリは、ユーザが定義した順序に従って並べられる。　NETCONF では、この順序は<edit-config>要求で特別な XML 属性を使用して制御される。　詳細はセクション 7.7.9 を参照のこと。

7.7.8. XML エンコーディング規則

leaf-list・ノードは、一連の XML 要素としてエンコードされる。　各要素のローカル名は leaf-list の識別子であり、その名前空間はモジュールの XML 名前空間である (セクション 7.1.3 を参照)。

各 leaf-list エントリの値は、型に従って XML にエンコードされ、要素内の文字データとして送信される。

leaf-list エントリを表す XML 要素は、leaf-list が"ordered-by user"の場合、ユーザが指定した順序で現れなければならない。それ以外の場合、順序は実装に依存する。 leaf-list のエントリを表す XML 要素は、leaf-list が RPC やアクションの入出力パラメータを定義していない限り、兄弟 leaf-list の要素と交互にエントリしてもよい。 例についてはセクション 7.7.10 を参照のこと。

7.7.9. NETCONF<edit-config>操作

leaf-list エントリの XML 要素で"operation"属性を使用すると、leaf-list エントリを作成および削除できるが、変更はできない。

"ordered-by user" leaf-list では、YANG XML 名前空間の属性"insert"と"value" (セクション 5.3.1)を使用して、leaf-list のどこにエントリを挿入するかを制御できる。 これらは、新しい leaf-list・エントリを挿入する"create"操作中、または新しい leaf-list エントリを挿入したり既存の leaf-list・エントリを移動する"merge"または"replace"操作中に使用できる。

"insert"属性には、"first"、"last"、"before"、"after"の値を指定できる。 値が"before"または"after"の場合、"value"属性は、leaf-list 中の既存のエントリを指定するためにも使用されなければならない。

"insert"操作に"create"属性が存在しない場合、デフォルトで"last"に設定される。

"ordered-by user" leaf-list 内の複数のエントリが同じ<edit-config>要求で変更される場合、エントリは一度に一つずつ、要求内の XML 要素の順序で変更される。

<copy-config>や、"replace"操作で leaf-list 全体を扱う<edit-config>では、leaf-list の順序はリクエスト中の XML 要素の順序と同じである。

NETCONF サーバが<edit-config>要求を処理するとき、leaf-list ノードの procedure の要素は以下のようになる。

- 操作が"merge"または"replace"の場合、leaf-list エントリが存在しなければ作成される。
- 操作が"create"の場合、leaf-list エントリが存在しなければ作成される。 leaf-list エントリが既に存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、leaf-list が存在すれば、そのエントリは削除される。 leaf-list エントリが存在しない場合は、"data-missing"エラーが返される。

7.7.10. 使用例

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

対応する XML インスタンスの例:

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

デフォルトの<edit-config>を使用して、この list に新しい要素を作成する。

操作"merge":

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <allow-user>eric</allow-user>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

次の順序で並べられたユーザ leaf-list を指定する。

```
leaf-list cipher {
  type string;
  ordered-by user;
  description
    "A list of ciphers.";
}
```

次の例は、新しい暗号"blowfish-cbc"を"3des-cbc"の後に挿入するために使用される。

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <cipher nc:operation="create"
```

```
                    yang:insert="after"
                    yang:value="3des-cbc">blowfish-cbc</cipher>
            </ssh>
          </services>
        </system>
      </config>
    </edit-config>
  </rpc>
```

7.8. "list"ステートメント

"list"ステートメントは、スキーマツリー内の内部データノードを定義するために使用される。 list ノードは、データツリー内の複数のインスタンスに存在する場合がある。 このような各インスタンスは、list エントリと呼ばれる。 "list"ステートメントは、識別子である引数をとり、その後に詳細な list 情報を保持するサブステートメントのブロックが続く。

list エントリは、定義されている場合、list のキーの値によって一意に識別される。

7.8.1. list の Substatements

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
key	7.8.2	0..1
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
notification	7.16	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
unique	7.8.3	0..n

```
| uses          | 7.13    | 0..n        |
| when          | 7.21.5  | 0..1        |
+---------------+---------+-------------+
```

7.8.2. list の"key"ステートメント

"key"ステートメントは、list が設定を表す場合には必ず存在し、それ以外の場合には存在してもよい。このステートメントは、この list の一つ以上のリーフ識別子を空白で区切った list を指定する文字列を引数として取る。　リーフ識別子は"key"の中に 2 回以上現れてはならない。　そのような各リーフ識別子は、list の子リーフを参照しなければならない。　リーフは、list のサブステートメントで直接定義することも、list で使用されるグループ化で定義することもできる。

"key"で指定されたすべてのリーフの組み合わせ値は、list エントリを一意に識別するために使用される。　すべてのキーリーフは、list エントリが作成されるときに値を与えられなければならない。したがって、キーリーフまたはそのタイプのデフォルト値は無視される。　キーリーフ内の"mandatory"ステートメントはすべて無視される。

キーの一部であるリーフは、任意の組み込み型または派生型にすることができる。

list 内のすべてのキーリーフは、その"config"が list 自体と同じ値でなければならない。

キー文字列構文は、正式にはセクション 14 の規則"key-arg"で定義されている。

7.8.3. list の"unique"ステートメント

"unique"ステートメントは、有効な list・エントリに制約を設定するために使用する。　これは、空白で区切られたスキーマノード識別子の list を含む文字列を引数として取り、子孫ノードの形式で与えなければならない(セクション 14 の"子孫スキーマノード ID"を参照)。　そのような各スキーマノード識別子は、リーフを参照しなければならない。

参照されるリーフの 1 つが設定データを表す場合、参照されるリーフのすべてが設定データを表さなければならない。

"unique"の制約は、デフォルト値を持つリーフを含む、引数文字列で指定されたすべてのリーフインスタンスの結合値が、すべての参照リーフが存在するか、またはデフォルト値を持つすべての list エントリンスタンス内で一意でなければならないことを指定する。　この制約はセクション 8 の規則に従って強制される。

一意な文字列構文は、正式にはセクション 14 の"一意引数"という規則で定義されている。

7.8.3.1.　使用例

次の list を使用する。

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

```
}
```

次の構成が無効である:

```
<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>
```

"http"と"FTP"の list があるので、次の設定は有効である。　エントリには参照されるすべてのリーフの値がないため、"unique"コンテナの適用時に考慮される。

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

7.8.4. list の子ノードステートメント

list 内では、"container"、"leaf"、"list"、"leaf-list"、"uses"、"choice"、"anydata"、"anyxml"ステートメントを使用して、list の子ノードを定義できる。

7.8.5. XML エンコーディング規則

list は一連の XML 要素としてエンコードされ、list 内のエントリごとに 1 つずつエンコードされる。　各要素のローカル名は list の識別子であり、その名前空間はモジュールの XML 名前空間である(セクション 7.1.3 を参照)。 list 全体を囲む XML 要素はない。

list のキーノードは、"key"ステートメント内で定義されているのと同じ順序で、list の identifier 要素のサブ要素としてエンコードされる。

list の残りの子ノードは、キーの後に、list 要素のサブ要素としてエンコードされる。　list で RPC またはアクションの入力または出力パラメータが定義されている場合、サブ要素は"list"ステート

メント内で定義されているのと同じ順序でエンコードされる。　それ以外の場合、サブ要素は任意の順序でエンコードされる。

list エントリのサブ要素間の空白は重要ではない。すなわち、実装はサブ要素間に空白文字を挿入してもよい。

list 項目を表す XML 要素は、list が"ordered-by user"の場合、ユーザが指定した順序で現れなければならない。それ以外の場合、順序は実装に依存する。　list エントリを表す XML 要素は、list が RPC またはアクションの入出力パラメータを定義していない限り、list の兄弟の要素とインターリーブしてもよい。

7.8.6. NETCONF<edit-config>操作

list のエントリは、list の XML 要素の"operation"属性を使用して、<edit-config>で作成、削除、置換、および変更できる。　いずれの場合も、すべてのキーの値を使用して list エントリを一意に識別する。　list エントリにすべてのキーが指定されていない場合は、"missing-element"エラーが返される。

"ordered-by user"list では、YANG XML 名前空間の属性"insert"と"key" (セクション 5.3.1)を使用して、list 内のどこにエントリを挿入するかを制御できる。　これらは、新しい list 項目を挿入する"create"操作の間、または新しい list 項目を挿入したり既存の list 項目を移動する"merge"または"replace"操作の間に使用できる。

"insert"属性には、"first"、"last"、"before"、"after"の値を指定できる。　値が"before"または"after"の場合、list 内の既存の要素を指定するために"key"属性も使用しなければならない。　"key"属性の値は、list 項目の完全なインスタンス ID の主要な述部である(セクション 9.13 参照)。

"insert"操作に"create"属性が存在しない場合、デフォルトで"last"に設定される。

"ordered-by user"list 内の複数のエントリが同じ<edit-config>要求で変更される場合、エントリは要求内の XML 要素の順序で一度に変更される。

list 全体をカバーする"replace"操作を持つ<copy-config>または<edit-config>では、list エントリの順序は要求内の XML 要素の順序と同じである。

NETCONF サーバが<edit-config>要求を処理する場合、list ノードの procedure の要素は以下のようになる。

- 操作が"merge"または"replace"の場合、list エントリが存在しなければ作成される。　list エントリがすでに存在し、"insert"および"key"属性が存在する場合、list エントリは"insert"および"key"属性の値に従って移動される。　list エントリが存在し、"insert"および"key"属性が存在しない場合、list エントリは移動されない。
- 操作が"create"の場合、list エントリが存在しなければ作成される。　list エントリが既に存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、エントリは存在すれば list から削除される。　list エントリが存在しない場合は、"data-missing"エラーが返される。

7.8.7. 使用例

次の list がある。

```
list user {
  key "name";
  config true;
  description
    "This is a list of users in the system.";
```

```
    leaf name {
      type string;
    }
    leaf type {
      type string;
    }
    leaf full-name {
      type string;
    }
  }
```

対応する XML インスタンスの例:

```
 <user>
    <name>fred</name>
    <type>admin</type>
    <full-name>Fred Flintstone</full-name>
  </user>
```

新規ユーザ作成"barney":

```
  <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
      <target>
        <running/>
      </target>
      <config>
        <system xmlns="urn:example:config">
          <user nc:operation="create">
            <name>barney</name>
            <type>admin</type>
            <full-name>Barney Rubble</full-name>
          </user>
        </system>
      </config>
    </edit-config>
  </rpc>
```

"fred"のタイプを"superuser"に変更するには、次の手順に従う。

```
  <rpc message-id="102"
```

```
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <user>
          <name>fred</name>
          <type>superuser</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

次の ordered-by user list を指定する。

```
list user {
  description
    "This is a list of users in the system.";
  ordered-by user;
  config true;
  key "first-name surname";
  leaf first-name {
    type string;
  }
  leaf surname {
    type string;
  }
  leaf type {
    type string;
  }
}
```

次の例は、新しいユーザ"barney rubble"をユーザ"fred flintstone"の後に挿入するために使用される。

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
```

```
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config"
           xmlns:ex="urn:example:config">
        <user nc:operation="create"
              yang:insert="after"
              yang:key="[ex:first-name='fred']
                        [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
          <type>admin</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

以下は、"barney rubble"前にユーザを移動するために使用される。

ユーザ"fred flintstone":

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config"
           xmlns:ex="urn:example:config">
        <user nc:operation="merge"
              yang:insert="before"
              yang:key="[ex:name='fred']
                        [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
        </user>
      </system>
    </config>
```

```
    </edit-config>
  </rpc>
```

7.9. "choice"ステートメント

"choice"ステートメントは choice の集合を定義し、その中の一つだけが任意の一つのデータツリーに存在することができる。　引数は識別子で、その後に詳細な choice 情報を保持するサブステートメントのブロックが続く。　この識別子は、スキーマツリー内の choice ノードを識別するために使用される。　データツリーに choice ノードがない。

choice は多くのブランチで構成され、各ブランチは"case"サブステートメントで定義される。　各ブランチには、多数の子ノードが含まれる。　choice のブランチのうち、最大で 1 つのブランチのノードが同時に存在する。

データツリーでは、choice した 1 つの case のみがいつでも有効であるため、1 つの case からノードを作成すると、他のすべての case からすべてのノードが暗黙的に削除される。　リクエストがある case からノードを作成する場合、サーバはその choice 内の他の case で定義されている既存のノードをすべて削除する。

7.9.1. choice のサブステートメント

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
default	7.9.3	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
mandatory	7.9.4	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.9.2. choice の"case"ステートメント

"case"ステートメントは、choice のブランチを定義するために使用される。引数として識別子をとり、その後に詳細な大文字小文字の情報を保持する副文のブロックを続ける。

識別子は、スキーマツリー内の case ノードを識別するために使用される。

case ノードがデータツリーに存在しない。

"case"ステートメント内では、"anydata"、 "anyxml"、"choice"、"container"、"leaf"、"list"、"leaf-list"、"uses"ステートメントを使用して、case ノードの子ノードを定義できる。これらすべての子ノードの識別子は、choice のすべての case において一意でなければならない。たとえば、次は不正である。

```
choice interface-type {     // This example is illegal YANG
  case a {
    leaf ethernet { ... }
  }
  case b {
    container ethernet { ...}
  }
}
```

省略形として、"case"ステートメントは、分岐に単一の"anydata"、"anyxml"、"choice"、"container"、"leaf"、"list"、"leaf-list"ステートメントが含まれる場合に省略できる。 この場合、case ノードはスキーマツリーに存在し、その識別子は子ノードの識別子と同じである。 スキーマノード識別子(セクション 6.5)は、常に明示的に case ノード識別子を含まなければならない。 次に例を示す。

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

は次と同じである。

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

case 識別子は choice 内で一意でなければならない。

7.9.2.1. case のサブステートメント

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n

```
| leaf          | 7.6     | 0..n        |
| leaf-list     | 7.7     | 0..n        |
| list          | 7.8     | 0..n        |
| reference     | 7.21.4  | 0..1        |
| status        | 7.21.2  | 0..1        |
| uses          | 7.13    | 0..n        |
| when          | 7.21.5  | 0..1        |
+---------------+---------+-------------+
```

7.9.3. choice の"default"ステートメント

"default"ステートメントは、choice のいずれの case にも子ノードが存在しない場合に、case を default と見なすかどうかを示す。　引数は、default の"case"ステートメントの識別子である。"default"ステートメントがない場合、default の大文字と小文字は区別されない。

"default"ステートメントは、"mandatory"が"true"である choice に存在してはならない。

default が重要なのは、case の下のノードの"default"ステートメントを考慮するときだけである(つまり、leaf と leaf-list のデフォルト値とネストされた choice の default)。　いずれの case の下にもノードが存在しない場合は、デフォルト値と default の下にネストされた default が使用される。

dafault の直下に必須ノード(セクション 3)があってはならない。

case の下の子ノードのデフォルト値は、その case の下のノードの 1 つが存在する場合、またはその case が dafault である場合にのみ使用される。　case の下にノードがなく、case が dafault でない場合、case の子ノードのデフォルト値は無視される。

この例では、choice のデフォルトは"interval"であり、"daily"、"time-of-day"、"manual"が存在しない場合はデフォルト値が使用される。　"daily"が存在する場合、"time-of-day"のデフォルト値が使用される。

```
container transfer {
  choice how {
    default interval;
    case interval {
      leaf interval {
        type uint16;
        units minutes;
        default 30;
      }
    }
    case daily {
      leaf daily {
        type empty;
      }
      leaf time-of-day {
        type string;
        units 24-hour-clock;
        default "01.00";
      }
    }
    case manual {
      leaf manual {
        type empty;
      }
```

```
        }
      }
    }
```

7.9.4. choice の"mandatory"ステートメント

"mandatory"ステートメントはオプションで、文字列"true"または"false"を引数として受け取り、有効なデータに制約を設定する。　もし"mandatory"が"true"なら、choice の case ブランチの一つから少なくとも一つのノードが存在しなければならない。　指定しない場合のデフォルトは"false"である。

制約の動作は、非存在コンテナ(セクション 7.5.1 を参照)ではないスキーマツリー内の choice で最も近い祖先ノードの型に依存する。

- スキーマツリーにそのような祖先ノードが存在しない場合、制約が適用される。
- そうでない場合、この祖先ノードが case ノードであれば、case ノードから他のノードが存在する場合に制約が適用される。
- それ以外の場合は、親ノードが存在する場合に強制される。

この制約はさらにセクション 8 の規則に従って強制される。

7.9.5. XML エンコーディング規則

choice ノードと case ノードは XML では表示されない。

"case"ステートメントの子ノードが RPC またはアクションの入力または出力パラメータ定義の一部である場合、それらは"case"ステートメントで定義されたのと同じ順序でエンコードされなければならない。　それ以外の場合、サブ要素は任意の順序でエンコードされる。

7.9.6. 使用例

次の choice がある。

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

対応する XML インスタンスの例:

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

プロトコルを TCP から UDP に変更するには、次の手順を実行する。

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <protocol>
          <udp nc:operation="create"/>
        </protocol>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.10. "anydata"ステートメント

"anydata"ステートメントは、スキーマツリーの内部ノードを定義する。これは、識別子である 1 つの引数と、その後に詳細な anydata 情報を保持する部分文のブロックをとる。

"anydata"ステートメントは、YANG でモデル化できるノードの未知のセットを表すために使用される。ただし、anyxml は除くが、モジュールの設計時にはデータモデルが不明である。必須ではないが、プロトコルシグナリングまたはこのドキュメントの範囲外のその他の手段によって、任意のデータコンテンツのデータモデルが既知になる可能性がある。

anydata が有用である例は、特定の通知が設計時に知られていない受信通知のリストである。anydata ノードは増やせません(セクション 7.17 参照)。

anydata ノードは、データツリーの 0 または 1 つのインスタンスに存在する。

実装は、anydata ノードの特定のインスタンスをモデル化するために使われるデータモデルを知っているかもしれないし、知らないかもしれない

anydata の使用は内容の操作を制限するので、"anydata"文は設定データを定義するために使われるべきではない[SHOULD NOT]。

7.10.1. anydata の副文

substatement	section	cardinality
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.10.2. XML エンコーディング規則

anydata ノードは XML 要素としてエンコードされる。要素のローカル名は anydata の識別子であり、名前空間はモジュールの XML 名前空間である (セクション 7.1.3 を参照)。 anydata ノードの値はノードのセットで、これは anydata 要素の XML サブ要素としてエンコードされる。

7.10.3. NETCONF<edit-config>操作

anydata ノードは不透明なデータの塊として扱われる。このデータは完全にのみ変更できる。

anydata ノードのサブ要素にある"操作"属性は、NETCONF サーバによって無視される。

NETCONF サーバが<edit-config>要求を処理する時、anydata ノードの手続きの要素は以下の通りである:

- 操作が"merge"または"replace"の場合、ノードが存在しない場合は作成され、その値は XML RPC データ内で見つかった anydata ノードのサブ要素に設定される。
- 操作が"create"の場合、ノードが存在しない場合は作成され、その値は XML RPC データ内で見つかった anydata ノードのサブ要素に設定される。ノードがすでに存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、ノードが存在すれば削除される。ノードが存在しない場合は、"data-missing"エラーが返される。

7.10.4. 使用例

次の"anydata"文がある。

```
list logged-notification {
  key time;
  leaf time {
    type yang:date-and-time;
  }
  anydata data;
}
```

このようなリストインスタンスの有効なエンコーディングを次に示す。

```
<logged-notification>
  <time>2014-07-29T13:43:12Z</time>
  <data>
    <notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
      <eventTime>2014-07-29T13:43:01Z</eventTime>
      <event xmlns="urn:example:event">
        <event-class>fault</event-class>
        <reporting-entity>
          <card>Ethernet0</card>
        </reporting-entity>
        <severity>major</severity>
      </event>
    </notification>
  </data>
</logged-notification>
```

7.11. "anyxml"ステートメント

"anyxml"ステートメントは、スキーマツリーの内部ノードを定義する。この関数は、1 つの引数（識別子）の後に、詳細な anyxml 情報を保持するサブステートメントのブロックを指定する。

"anyxml"ステートメントは、不明な XML の塊を表すために使用される。XML に制限はない。これは、RPC 応答などで便利である。例として、NETCONF の<get-config>操作の<filter>パラメータがある。

anyxml ノードは拡張できない(セクション 7.17 参照)。

anyxml ノードは、データツリーの 0 または 1 つのインスタンスに存在する。

anyxml を使用するとコンテンツの操作が制限されるため、設定データの定義に"anyxml"ステートメントを使用すべきではない[SHOULD NOT]。

YANG バージョン 1 では、"anyxml"が未知のデータ階層をモデル化できる唯一のステートメントであったことに注意すること。　多くの場合、この未知のデータ階層は実際に YANG でモデル化されるが、特定の YANG データモデルは設計時には知られていない。このような状況では、"anyxml"として扱われる"anydata"の代わりに(セクション 7.10)を使用することが推奨される[RECOMMENDED]。

7.11.1. anyxml のサブステートメント

substatement	section	cardinality
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1

```
| status          | 7.21.2  | 0..1        |
| when            | 7.21.5  | 0..1        |
+-----------------+---------+-------------+
```

7.11.2. XML エンコーディング規則

anyxml ノードは XML 要素としてエンコードされる。要素のローカル名は anyxml の識別子であり、その名前空間はモジュールの XML 名前空間である (セクション 7.1.3 を参照)。 anyxml ノードの値は、この要素の XML コンテンツとしてエンコードされる。

エンコーディングで使用される XML プレフィックスは、各インスタンスエンコーディングに対してローカルであることに注意すること。これは、同じ XML が異なる実装によって異なる方法でエンコードされる可能性があることを意味する。

7.11.3. NETCONF<edit-config>操作

anyxml ノードは不透明なデータの塊として扱われる。このデータは完全にのみ変更できる。

anyxml ノードのサブ要素に存在する"操作"属性は、NETCONF サーバによって無視される。

NETCONF サーバが<edit-config>要求を処理する場合、anyxml ノードの手順の要素は以下のようになる。

- 操作が"merge"または"replace"の場合、ノードが存在しない場合は作成され、その値は XML RPC データ内にある anyxml ノードの XML コンテンツに設定される。
- 操作が"create"の場合、ノードが存在しない場合は作成され、その値は XML RPC データで見つかった anyxml ノードの XML コンテンツに設定される。ノードがすでに存在する場合は、"data-exists"エラーが返される。
- 操作が"delete"の場合、ノードが存在すれば削除される。ノードが存在しない場合は、"data-missing"エラーが返される。

7.11.4. 使用例

次の"anyxml"文がある。

```
anyxml html-info;
```

次に、同じ anyxml 値の 2 つの有効なエンコーディングを示す。

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

7.12. "grouping"ステートメント

"grouping"ステートメントは、再利用可能なノードのブロックを定義するために使用され、セクション 5.5 の規則に従って、モジュールまたはサブモジュール、およびそこからインポートする他のモジュールでローカルに使用される。1 つの引数 (識別子) の後に、詳細なグループ化情報を保持するサブステートメントのブロックが続く。

"grouping"ステートメントはデータ定義ステートメントではないため、スキーマツリー内のノードを定義しない。

grouping は、従来のプログラミング言語における"structure"または"record"のようなものである。

grouping を定義すると、"uses"ステートメントで参照できる(セクション 7.13 参照)。grouping は、直接的にも間接的にも、他の grouping の連鎖を通して自身を参照してはならない(MUST NOT)。

grouping が YANG モジュールまたはサブモジュールのトップレベルで定義される場合、grouping の識別子はモジュール内で一意でなければならない(MUST)。

grouping は、単なるテキスト置換のメカニズムではない。ノードの集合も定義する。　グループ内に表示される識別子は、使用されている場所ではなく、グループが定義されているスコープに基づいて解決される。プレフィックスマッピング、タイプ名、グループ名、および拡張子の使用は、"grouping"ステートメントが表示される階層で評価される。拡張の場合、"grouping"ステートメントの直接の子として定義された拡張は、grouping 自体に適用される。

grouping で階層内にアクションまたは通知ノードが定義されている場合、すべてのコンテキストで使用することはできない。たとえば、rpc 定義では使用できない。セクション 7.15 及びセクション 7.16 参照。

7.12.1. grouping のサブステートメント

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.12.2. 使用例

```
import ietf-inet-types {
  prefix "inet";
}
grouping endpoint {
  description "A reusable endpoint group.";
  leaf ip {
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
}
```

7.13. "uses"ステートメント

"uses"ステートメントは、"grouping"定義を参照するために使用される。

引数は 1 つで、これはグループの名前である。

grouping への"uses"参照の効果は、grouping によって定義されたノードが現在のスキーマツリーにコピーされ、"refine"および"augment"ステートメントに従って更新されることである。

grouping で定義された識別子は、grouping の内容が"uses"ステートメント内に表示されない"grouping"ステートメントを介してスキーマツリーに追加されるまで名前空間にバインドされない。追加された時点で、現在のモジュールの名前空間にバインドされる。

7.13.1. uses のサブステートメント

substatement	section	cardinality
augment	7.17	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
refine	7.13.2	0..n
status	7.21.2	0..1
when	7.21.5	0..1

7.13.2. "refine"ステートメント

グループ内の各ノードのプロパティの一部は、"refine"ステートメントを使用して調整できる。引数は、グループ内のノードを識別する文字列である。このノードは、refine のターゲットノードと呼ばれる。grouping されたノードが"refine"ステートメントのターゲットノードとして存在しない場合、refine されないため、grouping で定義されたとおりに使用される。

引数の文字列は、子孫のスキーマノード識別子 (セクション 6.5 参照) である。次のリファインメントを実行できる。

- leaf ノードまたは choice ノードはデフォルト値を取得するか、すでにデフォルト値がある場合は新しいデフォルト値を取得する。
- leaf-list ノードは、デフォルト値のセット、またはデフォルト値がすでにある場合は新しいデフォルト値のセットを取得できる。すなわち、refine されたデフォルト値のセットは、既に与えられたデフォルトを置き換える。
- どのノードも特殊な"description"文字列を取得できる。
- どのノードも特殊な"reference"文字列を取得できる。
- どのノードも異なる"config"ステートメントを取得できる。
- leaf、anydata、anyxml、または choice ノードは、異なる"mandatory"ステートメントを取得できる。
- container ノードは"presence"ステートメントを取得する場合がある。
- leaf、leaf-list、list、container、anydata、anyxml ノードは、追加の"must"式を得ることができる。
- leaf-list または list ノードは、異なる"min-elements"または"max-elements"ステートメントを取得できる。
- leaf ノード、leaf-list ノード、list ノード、container ノード、choice ノード、case ノード、anydata ノード、anyxml ノードには、"if-feature"式が追加される。
- 拡張が細分化を許可する場合は、どのノードでも拡張を細分化できる。　詳細は 7.19 を参照のこと。

7.13.3. XML エンコーディング規則

grouping の各ノードは、別の XML 名前空間を持つ別のモジュールからインポートされた場合でも、インラインで定義されたかのようにエンコードされる。

7.13.4. 使用例

セクション 7.12.2 で定義されている"endpoint"グループ化を、他のモジュールの HTTP サーバの定義で使用するには、以下のようにする。

```
import example-system {
  prefix "sys";
}
container http-server {
  leaf name {
    type string;
  }
  uses sys:endpoint;
}
```

対応する XML インスタンスの例:

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

```
    <port>80</port>
  </http-server>
```

ポート 80 を HTTP サーバのデフォルトにする必要がある場合、デフォルトでは次のことが可能である。

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

サーバのリストを定義し、各サーバがキーとして"ip"と"port"を持っている場合、以下のことができる。

```
  list server {
    key "ip port";
    leaf name {
      type string;
    }
    uses sys:endpoint;
  }
```

以下はエラーである。

```
  container http-server {
    uses sys:endpoint;
    leaf ip {          // illegal - same identifier "ip" used twice
      type string;
    }
  }
```

7.14. "rpc"ステートメント

"rpc"ステートメントは、RPC 操作を定義するために使用する。1 つの引数 (識別子) の後に、詳細な rpc 情報を保持するサブステートメントのブロックが続く。この引数は RPC の名前である。

"rpc"ステートメントは、スキーマツリーに rpc ノードを定義する。rpc ノードの下には、"input"という名前のスキーマノードと"output"という名前のスキーマノードも定義される。　ノード"input"と"output"は、モジュールの名前空間で定義される。

7.14.1. RPC のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n

7.14.2. "input"ステートメント

"input"文はオプションで、操作の input パラメータを定義するために使用する。そのことは議論の余地はない。"input"サブステートメントは、操作の input ノードの下にノードを定義する。

input ツリーの leaf が値"mandatory"を持つ"true"ステートメントを持つ場合、その leaf は RPC 呼び出しに存在しなければならない(MUST)。

もし入力ツリーのリーフがデフォルト値を持つなら、サーバは、セクション 7.6.1 に記述されているのと同じ場合に、この値を使用しなければならない(MUST)。このような場合、サーバは、デフォルト値を値として RPC 呼び出しに leaf が存在するかのように動作しなければならない(MUST)。

input ツリー中の leaf-list が一つ以上のデフォルト値を持つ場合、サーバは、セクション 7.7.2 に記述されている場合と同じ場合にこれらの値を使用しなければならない(MUST)。このような場合、サーバは RPC 呼び出しに leaf-list が存在し、その値がデフォルト値であるかのように動作しなければならない(MUST)。

input ツリーはデータストアの一部ではないため、input ツリー内のノードの"config"ステートメントはすべて無視される。

ノードが"when"ステートメントを持ち、それが"false"と評価される場合、このノードは input ツリーに存在してはならない(MUST NOT)。

7.14.2.1. input のサブステートメント

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n

must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.3. "output"ステートメント

"output"ステートメントは、RPC オペレーションの出力パラメータを定義するために使用される。そのことは議論の余地はない。"output"サブステートメントは、操作の output ノードの下にノードを定義する。

output ツリーのリーフが値"mandatory"を持つ"true"ステートメントを持つ場合、その leaf は RPC 応答に存在しなければならない(MUST)。

output ツリーの leaf がデフォルト値を持つ場合、クライアントは、セクション 7.6.1 に記述されている場合と同じ場合にこの値を使用しなければならない(MUST)。このような場合、クライアントは、デフォルト値を値として RPC 応答に leaf が存在するかのように動作しなければならない(MUST)。

output ツリー中の leaf-list が一つ以上のデフォルト値を持つ場合、クライアントはセクション 7.7.2 に記述されている場合と同じ場合にこれらの値を使用しなければならない(MUST)。このような場合、クライアントは、RPC 応答に leaf-list が存在し、その値がデフォルト値であるかのように動作しなければならない(MUST)。

output ツリーはデータストアの一部ではないため、output ツリー内のノードの"config"ステートメントはすべて無視される。

ノードが"when"ステートメントを持ち、それが"false"と評価される場合、このノードは output ツリーに存在してはならない(MUST NOT)。

7.14.3.1. output の Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.4. NETCONF XML エンコーディング規則

rpc ノードは<rpc>要素の子 XML 要素として符号化される。また、[RFC 6241]で代理グループ"rpcOperation"によって記述される。要素のローカル名は rpc の識別子であり、名前空間はモジュールの XML 名前空間である(セクション 7.1.3 を参照。)。

input パラメータは、rpc ノードの XML 要素の子 XML 要素として、"input"ステートメントで定義されているのと同じ順序でエンコードされる。

RPC オペレーションの呼び出しが成功し、output パラメータが返されなかった場合、<rpc-reply>は[RFC 6241]で定義された 1 つの<ok/>要素を含む。もし、output パラメータが返された場合、それらは [RFC 6241]で定義された<rpc-reply>要素に対し、子要素として、"output"ステートメント内で定義されている順序通りに、エンコードされる。

7.14.5. 使用例

次に、RPC 動作を定義する例を示す。

```
module example-rock {
  yang-version 1.1;
  namespace "urn:example:rock";
  prefix "rock";
  rpc rock-the-house {
    input {
      leaf zip-code {
        type string;
      }
    }
  }
}
```

完全な rpc および rpc-reply の対応する XML インスタンスの例:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="urn:example:rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>
<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

7.15. "action"ステートメント

"action"ステートメントは、特定のコンテナまたはリストデータノードに接続された操作を定義するために使用される。1 つの引数 (識別子) の後に、詳細なアクション情報を保持するサブステートメントのブロックが続く。引数はアクションの名前である。

"action"ステートメントは、スキーマツリー内の action ノードを定義する。action ノードの下には、"input"という名前のスキーマノードと"output"という名前のスキーマノードも定義されている。ノード"input"と"output"は、モジュールの名前空間で定義される。

action は、rpc、別の action、または notification 内で定義してはならない(MUST NOT)。すなわち、action ノードは、rpc、action、または notification ノードをスキーマツリー内の上位ノー

ドの 1 つとして持ってはならない(MUST NOT)。たとえば、ノード階層のどこかに action を含むグループが notification 定義で使用されている場合、これはエラーであることを意味する。

action は、"key"ステートメントのない list ノードである祖先ノードを持ってはならない(MUST NOT)。

action はモジュールの最上位レベルまたは"case"ステートメントでは定義できないため、ノード階層の最上位にアクションを含むグループ化がモジュールの最上位レベルまたは case 定義で使用されると、エラーになる。

action と rpc の違いは、action がデータストア内のノードに関連付けられているのに対し、rpc は関連付けられていないことである。action が呼び出されると、データストア内のノードが action 名と input パラメータとともに指定される。

7.15.1. action のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n

7.15.2. NETCONF XML エンコーディング規則

action が起動されると、名前空間"urn:ietf:params:xml:ns:yang:1"(セクション 5.3.1 を参照)内のローカル名"action"を持つ要素は、 [RFC 6241]で定義される"rpc"要素に対する子 XML 要素としてエンコードされ、[RFC 6241]の代理グループ"rpcOperation"によって記述される。

<action>要素には、データストア内のノードを識別するノードの階層が含まれる。これは、トップレベルからアクションを含むリストまたはコンテナまでの直接パス内のすべてのコンテナとリストノードを含まなければならない(MUST)。リストについては、すべてのキーリーフも含めなければならない(MUST)。最も内側のコンテナーまたはリストには、定義されたアクションの名前を持つ XML 要素が含まれる　この要素内では、入力パラメーターは、"入力"ステートメント内で定義されるのと同じ順序で、子 XML 要素としてエンコードされる。

1 つの<rpc>で起動できるアクションは 1 つだけである <rpc>に複数のアクションがある場合、サーバは"不良素子"<error-tag>を<rpc-error>に含めて応答しなければならない(MUST)。

アクション操作の呼び出しが成功し、出力パラメータが返されなかった場合、<rpc-reply>は [RFC 6241]． 出力パラメータが返された場合、それらは [RFC 6241]は、"output"ステートメント内で定義されている順序と同じである。

7.15.3. 使用例

次の例では、サーバファームの 1 つのサーバをリセットするアクションを定義する。

```
module example-server-farm {
```

```
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";
  import ietf-yang-types {
    prefix "yang";
  }
  list server {
    key name;
    leaf name {
      type string;
    }
    action reset {
      input {
        leaf reset-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

完全な rpc および rpc-reply の対応する XML インスタンスの例:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <server xmlns="urn:example:server-farm">
      <name>apache-1</name>
      <reset>
        <reset-at>2014-07-29T13:42:00Z</reset-at>
      </reset>
    </server>
  </action>
</rpc>
<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
      <reset-finished-at xmlns="urn:example:server-farm">
        2014-07-29T13:42:12Z
      </reset-finished-at>
    </rpc-reply>
```

7.16. "notification"ステートメント

"notification"ステートメントは、notification を定義するために使用される。 1 つの引数(識別子) の後に、詳細な notification 情報を保持するサブステートメントのブロックが続く。"notification"ステートメントは、スキーマツリーに notification ノードを定義する。

notification は、モジュールの最上位で定義するか、スキーマツリー内の特定のコンテナまたはリストデータノードに接続できる。

notification は、rpc、action、または他の notification 内で定義されてはならない(MUST NOT)。すなわち、notification ノードは、rpc、action、または notification ノードをスキーマツリー内の上位ノードの 1 つとして持ってはならない(MUST NOT)。たとえば、ノード階層のどこかに notification を含むグループ化が rpc 定義で使用されている場合は、エラーになる。

notification は"key"文のない list ノードである祖先ノードを持ってはならない(MUST NOT)。

notification は"case"ステートメントでは定義できないため、notification をノード階層の最上位に含むグループがケース定義で使用されている場合はエラーになる。

notification ツリー内の leaf が値"mandatory"を持つ"true"ステートメントを持つ場合、その leaf は notification インスタンス内に存在しなければならない(MUST)。

もし notification ツリーの leaf がデフォルト値を持つなら、クライアントはセクション 7.6.1 に記述されているのと同じ場合にこの値を使用しなければならない(MUST)。このような場合、クライアントは、デフォルト値を値として notification インスタンスに leaf が存在するかのように動作しなければならない(MUST)。

もし notification ツリーの中の leaf-list が一つ以上のデフォルト値を持つなら、クライアントはセクション 7.7.2 に記述されているのと同じ場合にこれらの値を使わなければならない(MUST)。このような場合、クライアントは、leaf-list がその値としてデフォルト値を持つ notification インスタンスに存在するかのように動作しなければならない。(MUST)

notification ツリーはデータストアの一部ではないため、notification ツリー内のノードの"設定"ステートメントはすべて無視される。

7.16.1. notification の Substatements

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n

```
| must          | 7.5.3    | 0..n         |
| reference     | 7.21.4   | 0..1         |
| status        | 7.21.2   | 0..1         |
| typedef       | 7.3      | 0..n         |
| uses          | 7.13     | 0..n         |
+---------------+----------+--------------+
```

7.16.2. NETCONF XML エンコーディング規則

モジュールのトップレベルで定義された notification ノードは、"NETCONF イベント notification" [RFC 5277]. 要素のローカル名は notification の識別子であり、その名前空間はモジュールの XML 名前空間(セクション 7.1.3 を参照)である。

notification ノードがデータノードの子として定義されている場合、[RFC 5277]の <notification>要素は、データストア内のノードを識別するノードの階層が含まれる。 トップレベルから notification を含むリストまたはコンテナまで、すべてのコンテナとリストノードを含まなければならない(MUST)。リストについては、すべての key leaf も含めなければならない(MUST)。最も内側のコンテナーまたはリストには、定義された notification の名前を持つ XML 要素が含まれる。

notification の子ノードは、notification ノードの XML 要素のサブ要素として任意の順序でエンコードされる。

7.16.3. 使用例

次の例では、モジュールのトップレベルで notification を定義する。

```
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;
    }
  }
}
```

完全な notification の対応する XML インスタンスの例:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <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>
```

次の例では、データノードに notification を定義する。

```
  module example-interface-module {
    yang-version 1.1;
    namespace "urn:example:interface-module";
    prefix "if";
    container interfaces {
      list interface {
        key "name";
        leaf name {
          type string;
        }
        notification interface-enabled {
          leaf by-user {
            type string;
          }
        }
      }
    }
  }
```

完全な notification の対応する XML インスタンスの例:

```
  <notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2008-07-08T00:01:00Z</eventTime>
    <interfaces xmlns="urn:example:interface-module">
      <interface>
        <name>eth1</name>
        <interface-enabled>
          <by-user>fred</by-user>
        </interface-enabled>
      </interface>
    </interfaces>
```

```
    </notification>
```

7.17. "augment"ステートメント

"augment"ステートメントを使用すると、モジュールまたはサブモジュールを、外部モジュールまたは現在のモジュールとそのサブモジュールで定義されているスキーマツリーに追加したり、"uses"ステートメント内のグループからノードに追加したりできる。引数は、スキーマツリー内のノードを識別する文字列である。このノードは、augment のターゲットノードと呼ばれる。ターゲットノードは、container、list、choice、case、input、output、または notification ノードのいずれかでなければならない(MUST)。これに、"augment"ステートメントに続くサブステートメントで定義されたノードが追加される。

引数の文字列はスキーマノード識別子(セクション 6.5 参照)である。"augment"ステートメントがモジュールまたはサブモジュールのトップレベルにある場合、スキーマノード識別子の絶対形式(セクション 14 の"絶対スキーマノード ID"で規定)を使用しなければならない(MUST)。"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である場合、子孫形式(セクション 14 の"子孫スキーマノード ID"で規定)を使用しなければならない(MUST)。

ターゲットノードが container、list、case、input、output、または notification ノードの場合、"container"、"leaf"、"list"、"leaf-list"、"uses"、および"choice"ステートメントを"augment"ステートメント内で使用できる。

ターゲットノードが container または list ノードの場合、"action"ステートメントと"notification"ステートメントを"augment"ステートメント内で使用できる。

ターゲットノードが choice ノードである場合、"case"ステートメントまたは省略された"case"ステートメント(セクション 7.9.2 参照)を"augment"ステートメント内で使用できる。

"augment"ステートメントは、同じモジュールからターゲットノードに同じ名前の複数のノードを追加してはならない(MUST NOT)。

もし augment による追加が、設定を表す必須ノード(セクション 3 参照)を他のモジュールのターゲットノードに追加するなら、拡張は"when"ステートメントを用いて条件付きにしなければならない(MUST)。"when"式を定義するときは、拡張モジュールのことを知らないクライアントが破壊されないように注意しなければならない。

次の例では、"interface"エントリに"mandatory-leaf"を追加してもかまわない。これは、この追加が"some-new-iftype"のサポートに依存しているためである。古いクライアントはこの型を知らないので、この型を選択することはなく、したがって必須のデータノードを追加することはない。

```
module example-augment {
  yang-version 1.1;
  namespace "urn:example:augment";
  prefix mymod;
  import ietf-interfaces {
    prefix if;
  }
  identity some-new-iftype {
     base if:interface-type;
  }
  augment "/if:interfaces/if:interface" {
     when 'derived-from-or-self(if:type, "mymod:some-new-iftype")';
     leaf mandatory-leaf {
       mandatory true;
       type string;
```

```
        }
      }
    }
```

7.17.1. augment のサブステートメント

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
uses	7.13	0..n
when	7.21.5	0..1

7.17.2. XML エンコーディング規則

"augment"ステートメントで定義されたすべてのデータノードは、"augment"が指定されたモジュールの XML 名前空間で XML 要素として定義される。

ノードが拡張されるとき、拡張される子ノードは、任意の順序で、拡張されるノードへの副要素として符号化される。

7.17.3. 使用例

namespace urn:example:interface-module には、次のものがある。

```
  container interfaces {
    list ifEntry {
      key "ifIndex";
      leaf ifIndex {
        type uint32;
      }
      leaf ifDescr {
```

```
        type string;
      }
      leaf ifType {
        type iana:IfType;
      }
      leaf ifMtu {
        type int32;
      }
    }
  }
```

名前空間 urn:example:ds0 では、次のようになる。

```
  import example-interface-module {
    prefix "if";
  }
  augment "/if:interfaces/if:ifEntry" {
    when "if:ifType='ds0'";
    leaf ds0ChannelNumber {
      type ChannelNumber;
    }
  }
```

対応する XML インスタンスの例:

```
 <interfaces xmlns="urn:example:interface-module"
             xmlns:ds0="urn:example:ds0">
    <ifEntry>
      <ifIndex>1</ifIndex>
      <ifDescr>Flintstone Inc Ethernet A562</ifDescr>
      <ifType>ethernetCsmacd</ifType>
      <ifMtu>1500</ifMtu>
    </ifEntry>
    <ifEntry>
      <ifIndex>2</ifIndex>
      <ifDescr>Flintstone Inc DS0</ifDescr>
      <ifType>ds0</ifType>
      <ds0:ds0ChannelNumber>1</ds0:ds0ChannelNumber>
    </ifEntry>
  </interfaces>
```

別の例として、セクション 7.9.6 で定義されている choice があるとする。次の構文を使用して、プロトコル定義を拡張できる。

```
augment /ex:system/ex:protocol/ex:name {
  case c {
    leaf smtp {
      type empty;
    }
  }
}
```

対応する XML インスタンスの例:

```
<ex:system>
  <ex:protocol>
    <ex:tcp/>
  </ex:protocol>
</ex:system>
```

または

```
<ex:system>
  <ex:protocol>
    <other:smtp/>
  </ex:protocol>
</ex:system>
```

7.18. "identity"ステートメント

"identity"ステートメントは、グローバルに一意、抽象、および型指定なしの新しい ID を定義するために使用される。ID の唯一の目的は、名前、セマンティクス、存在を示すことである。ID は、一から定義することも、1 つ以上の基本 ID から派生させることもできる。identity の引数は、名前となる識別子である。その後に、詳細な ID 情報を保持するサブステートメントのブロックが続きます。

組み込みデータ型"identityref" (セクション 9.10 参照)を使用して、データモデル内の ID を参照できる。

7.18.1. identity のサブステートメント

substatement	section	cardinality
base	7.18.2	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1

```
+--------------+---------+-------------+
```

7.18.2. "base"ステートメント

"base"ステートメントは、引数として、新しいIDの派生元となる既存IDの名前を指す文字列を取る。"base"ステートメントが存在しない場合、IDは一から定義される。複数の"base"ステートメントが存在する場合、IDはそれらすべてから派生する。

baseの名前にプレフィクスが存在する場合は、そのプレフィクスを使用してインポートされたモジュールで定義されているIDを指し、プレフィクスがローカルモジュールのプレフィクスと一致する場合はローカルモジュールを指す。そうでなければ、名前が一致するIDを現在のモジュールまたは含まれるサブモジュールで定義しなければならない(MUST)。

IDは、直接的にも間接的にも、自身を参照してはならない(MUST NOT)。

IDの派生には、次の特性がある。

- 再帰的ではない。つまり、IDはそれ自体から派生したものではないということである。
- 推移的である。つまり、ID BがAから派生し、ID CがBから派生する場合、ID CもAから派生する。

7.18.3. 使用例

```
module example-crypto-base {
  yang-version 1.1;
  namespace "urn:example:crypto-base";
  prefix "crypto";
  identity crypto-alg {
    description
      "Base identity from which all crypto algorithms
       are derived.";
  }
  identity symmetric-key {
    description
      "Base identity used to identify symmetric-key crypto
       algorithms.";
    }
  identity public-key {
    description
      "Base identity used to identify public-key crypto
       algorithms.";
    }
}
module example-des {
  yang-version 1.1;
  namespace "urn:example:des";
  prefix "des";
  import "example-crypto-base" {
    prefix "crypto";
```

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

7.19. "extension"ステートメント

"extension"ステートメントを使用すると、YANG 言語内で新しいステートメントを定義できる。この新しいステートメント定義は、他のモジュールにインポートして使用できる。

"extension"ステートメントの引数は、拡張の新しいキーワードである識別子であり、その後に詳細な拡張情報を保持するサブステートメントのブロックが続く必要がある。"extension"ステートメントの目的は、他のモジュールがインポートして使用できるようにキーワードを定義することである。

この拡張は、通常の YANG ステートメントと同様に使用でき、"extension"ステートメントが定義されている場合はステートメント名の後に、引数、およびオプションのサブステートメントのブロックが続く。ステートメントの名前は、extension が定義されたモジュールの接頭辞、コロン (":") 、および extension のキーワードを、空白を入れずに組み合わせて作成される。extension のサブステートメントは、"extension"ステートメントによって定義され、この仕様の範囲外のメカニズムを使用する。構文上、サブステートメントは、"extension"ステートメントを使用して定義された拡張を含む YANG ステートメントでなければならない(MUST)。拡張における YANG ステートメントは、セクション 14 の構文規則に従わなければならない(MUST)。

extension は洗練(セクション 7.13.2 参照)と逸脱(セクション 7.20.3.2 参照)を許すことができるが、これがどのように定義されるかのメカニズムはこの仕様の範囲外である。

7.19.1. extension のサブステートメント

substatement	section	cardinality
argument	7.19.2	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1

7.19.2. "argument"ステートメント

"argument"ステートメントはオプションで、キーワードの引数の名前である文字列を引数として取る。"argument"ステートメントが存在しない場合、キーワードは使用時に引数を必要としない。

引数の名前はYINマッピングで使用され、引数の"yin-element"ステートメントに応じてXML属性または要素名として使用される。

7.19.2.1. argumentのサブステートメント

substatement	section	cardinality
yin-element	7.19.2.2	0..1

7.19.2.2. "yin-element"ステートメント

"yin-element"ステートメントはオプションで、文字列"true"または"false"を引数として取る。このステートメントは、引数がYINのXMLエレメントにマップされているか、XML属性(セクション13参照)にマップされているかを示す。"yin-element"ステートメントがない場合は、デフォルトで"false"に設定される。

7.19.3. 使用例

拡張を定義する手順は、次のとおりである。

```
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";
  }
}
```

拡張子を使用するには、次の手順に従う。

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

```
      myext:c-define "MY_INTERFACES";
    }
  }
```

7.20. 適合性関連ステートメント

この節は、セクション 5.6 に記述されるように、適合性に関連するステートメントを定義する。

7.20.1. "feature"ステートメント

"feature"ステートメントは、スキーマの一部を条件付きとしてマークするメカニズムを定義するために使用される。feature の名前を定義しておき、その後、"if-feature"ステートメントを使用して参照できる(セクション 7.20.2 参照)。"if-feature"ステートメントでタグ付けされたスキーマノードは、サーバが指定された feature 式をサポートしていない限り、サーバによって無視される。これにより、YANG モジュールの一部をサーバの条件に基づいて条件付きにすることができる。このモデルは、モデル内のサーバの能力を表すことができ、サーバの能力や役割を変えることができる、より豊富なモデルを提供する。

"feature"ステートメントへの引数は、新しい feature の名前であり、セクション 6.2 の識別子に関する規則に従う。この名前は、スキーマノードを feature に関連付けるために"if-feature"ステートメントで使用される。

この例では、"local-storage"という機能は、サーバが何らかのローカルストレージに syslog メッセージを保存する機能を表す。この機能は、何らかのローカルストレージの存在を"local-storage-limit"リーフの条件にするために使用される。サーバがこの機能をサポートしていることを報告しない場合、"local-storage-limit"ノードはサポートされない。

```
  module example-syslog {
    yang-version 1.1;
    ...
    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"ステートメントは、YANG 構文内の多くの場所で使用できる。"if-feature"のタグが付いた定義は、サーバがその機能をサポートしていない場合は無視される。

feature は、直接的または間接的に自身を参照してはならない(MUST NOT)。

サーバが他の機能に依存する機能をサポートするためには、サーバはすべての依存する機能もサポートしなければならない(MUST)。faeture は、ひとつまたは複数の"if-feature"サブステートメントを持つことがある。

7.20.1.1. feature のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1

7.20.2. "if-feature"ステートメント

"if-feature"ステートメントは、その親ステートメントを条件付きにする。引数は、feature 名に対する bool 式である。この式では、feature がサーバでサポートされている場合に限り、feature 名は"true"と評価される。親ステートメントは、bool 式が"true"と評価されるサーバによって実装される。

if-feature の bool 式構文は、セクション 14 の規則"if-feature-expr 関数"で正式に定義されている。括弧は式をグループ化するために使用される。式が評価される場合、優先順位は、括弧によるグループ化 "()" 、"not"、"and"、"or"の順番になる。

bool 式の feature 名にプレフィクスが存在する場合、プレフィクス付きの名前は、そのプレフィクスを使用してインポートされたモジュールで定義された feature、またはプレフィクスがローカルモジュールのプレフィクスと一致する場合はローカルモジュールを参照する。そうでない場合、名前が一致する feature は、現在のモジュールまたは含まれるサブモジュールで定義されなければならない(MUST)。リストキーである leaf は、いかなる"if-feature"ステートメントも持ってはならない(MUST NOT)。

7.20.2.1. 使用例

この例では、コンテナ"target"は、"outbound-tls"または"outbound-ssh"のいずれかの feature がサーバによってサポートされている場合に実装される。

```
container target {
  if-feature "outbound-tls or outbound-ssh";
  ...
}
```

次の例は同じである。

```
if-feature "not foo or ar and baz";
if-feature "(not foo) or (bar and baz)";
```

7.20.3. "deviation"ステートメント

"deviation"ステートメントは、サーバが忠実に実装していないモジュールの階層を定義する。引数は、モジュールからの逸脱が発生するスキーマツリー内のノードを識別する文字列である。このノードは、逸脱のターゲットノードと呼ばれる。"deviation"ステートメントの内容は、逸脱の詳細を示す。
引数の文字列は絶対スキーマノード識別子(セクション 6.5 参照)である。

deviation は、サーバまたはサーバクラスが標準から逸脱する方法を定義する。これは、deviation は、実装が標準からどのように変化するかを学習するためのメカニズムであるため、決して公開された標準の一部であってはならないことを意味する(MUST NOT)。

サーバの逸脱は強く非推奨であり、最後の手段としてのみ使用しなければならない(MUST)。サーバが標準に従っていないことをアプリケーションに伝えることは、標準を正しく実装することに代わるものではない。モジュールから外れたサーバは、モジュールに完全に準拠していない。

しかし、場合によっては、特定のデバイスが標準モジュールの部品をサポートするハードウェアまたはソフトウェアの能力を持たないことがある。このような場合、サーバは、モジュールのサポートされていない部分を設定しようとする試みをエラーとして扱い、そのエラーを疑うことを知らないアプリケーションに報告するか、またはそれらの着信要求を無視するかを選択する。どちらの選択も受け入れられない。

代わりに、YANG では、"deviation"ステートメントを使用して、サポートされていないベースモジュールの部分、またはサポートされているが異なる構文を持つベースモジュールの部分をサーバが文書化できる。

サーバによって公表されたすべての deviation を適用した後、結果のデータモデルは、どのような順序であっても、依然として有効でなければならならない(MUST)。

7.20.3.1. deviation のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
deviate	7.20.3.2	1..n
reference	7.21.4	0..1

7.20.3.2. "deviate"ステートメント

"deviate"ステートメントは、サーバによるターゲットノードの実装が元の定義とどのように異なるかを定義する。引数は、"not-supported"、"add"、"replace"、"delete"のいずれかの文字列である。

引数"not-supported"は、ターゲットノードがこのサーバによって実装されていないことを示す。

引数"add"は、ターゲットノードにプロパティを追加する。追加するプロパティは、"deviate"ステートメントのサブステートメントで識別される。プロパティが一度しか出現できない場合、そのプロパティはターゲットノードに存在してはならない(MUST NOT)。

引数"replace"は、ターゲットノードのプロパティを置き換える。置き換えるプロパティは、"deviate"ステートメントのサブステートメントで識別される。置き換えるプロパティは、ターゲットノードに存在している必要がある(MUST)。

引数"delete"は、ターゲットノードからプロパティを削除する。削除するプロパティは、"delete"ステートメントのサブステートメントによって識別される。サブステートメントのキーワードは、ターゲットノードの対応するキーワードと一致する必要があり、引数の文字列は、ターゲットノードの対応するキーワードの引数文字列と等しい必要がある(MUST)。

substatement	section	cardinality
config	7.21.1	0..1
default	7.6.4, 7.7.4	0..n
mandatory	7.6.5	0..1
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
type	7.4	0..1
unique	7.8.3	0..n
units	7.3.3	0..1

deviate のサブステートメント

ターゲットノードに extension で定義されたプロパティがある場合、extension で deviation が許容されると、このプロパティは逸脱する可能性がある。詳細はセクション 7.19 を参照のこと。

7.20.3.3. 使用例

この例では、サーバはクライアントアプリケーションに対して、RFC 867 で定義されるスタイルの "daytime"サービスをサポートしていないことを通知している。

```
module example-deviations {
  yang-version 1.1;
  namespace "urn:example:deviations";
  prefix md;
  import example-base {
    prefix base;
  }
  deviation /base:system/base:daytime {
    deviate not-supported;
  }
  ...
}
```

サーバはモジュール"example-base"と"example-deviations"の両方を通知する。

次の例では、サーバ固有のデフォルト値を、デフォルト値が定義されていない leaf に設定する。

```
deviation /base:system/base:user/base:type {
  deviate add {
    default "admin"; // new users are 'admin' by default
  }
```

```
}
```

この例では、サーバはネームサーバの数を 3 に制限する。

```
deviation /base:system/base:name-server {
  deviate replace {
    max-elements 3;
  }
}
```

元の定義が次のような場合、

```
container system {
  must "daytime or time";
  ...      }
```

サーバは次のようにしてこの"must"制約を取り除くことができる。

```
deviation /base:system {
  deviate delete {
    must "daytime or time";
  }
}
```

7.21. 共通ステートメント

このセクションでは、他のいくつかのステートメントに共通するサブステートメントを定義する。

7.21.1. "config"ステートメント

"config"ステートメントは、文字列"true"または"false"を引数として取る。"config"が"true"の場合、定義は構成を表す。構成を表すデータノードは、構成データストアの一部である。

"config"が"false"の場合、定義は状態データを表す。状態データを表すデータノードは、構成データストアの一部ではない。

"config"を指定しない場合、デフォルトは親スキーマノードの"config"値と同じになる。親ノードが case ノードの場合、この値は case ノードの親である choice ノードと同じである。

最上位ノードが"config"ステートメントを指定しない場合、デフォルトは"true"である。

ノードの"config"が"false"に設定されている場合、その下にあるノードの"config"を"true"に設定することはできない(MUST NOT)。

7.21.2. "status"ステートメント

"status"ステートメントは、文字列"current"、"deprecated"、または"obsolete"のいずれかを引数として取る。

o "current"は、定義が最新で有効であることを意味する。

- "deprecated"は古い定義を示すが、古い/既存の実装との相互運用性を促進するために、新しい/継続的な実装を許可する。
- "obsolete"とは、定義が時代遅れであり、実装すべきではないこと、および/または実装から削除できることを意味する。ステータスを指定しない場合、デフォルトは"current"である。

定義が"current"の場合、同じモジュール内の"deprecated"または"obsolete"定義を参照してはならない(MUST NOT)。

定義が"deprecated"の場合、同じモジュール内の"obsolete"定義を参照してはならない(MUST NOT)。

たとえば、以下は不正である。

```
typedef my-type {
  status deprecated;
  type int32;
}
leaf my-leaf {
  status current;
  type my-type; // illegal, since my-type is deprecated
}
```

7.21.3. "description"ステートメント

"description"ステートメントは、人間が読むことが可能なテキストの説明文を含む文字列を引数として取る。テキストは、モジュール開発者が選択した単一または複数の言語で提供される。相互運用性のために、モジュールを使用するネットワーク管理者のコミュニティで広く理解されている言語を選択することが推奨される(RECOMMENDED)。

7.21.4. "reference"ステートメント

"reference"ステートメントは、関連する管理情報を定義する別のモジュール、またはこの定義に関連する追加情報を提供するドキュメントのいずれかの外部ドキュメントを表す、人間が読み取り可能な相互参照である文字列を引数として取る。

たとえば、"uri"データ型の typedef は次のようになる。

```
typedef uri {
  type string;
  reference
    "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
  ...
}
```

7.21.5. "when"ステートメント

"when"ステートメントは、その親データ定義ステートメントを条件付きにする。親データ定義ステートメントで定義されたノードは、"when"ステートメントで指定された条件が満たされる場合にのみ有効である。ステートメントの引数は、この条件を正式に指定するために使用される XPath 式(セクション 6.4 参照)である。XPath 式が概念的に特定のインスタンスに対して"true"と評価される場合、親デー

タ定義ステートメントによって定義されたノードは有効である。それ以外の場合はそうではない。リストキーである leaf は、"when"ステートメントを持ってはならない(MUST NOT)。

リストで使用されるグループ化でキーリーフが定義されている場合、"uses"ステートメントは"when"ステートメントを持ってはならない(MUST NOT)。

詳細はセクション 8.3.2 を参照のこと。

XPath 式は、セクション 6.4.1 の定義に加えて、次のコンテキストで概念的に評価される。

- "when"ステートメントが"augment"ステートメントの子である場合、ターゲットノードがデータノードであれば、コンテキストノードはデータツリー内の augment のターゲットノードになる。それ以外の場合、コンテキストノードは、データノードでもあるターゲットノードに最も近い祖先ノードである。このようなノードが存在しない場合、コンテキストノードはルートノードになる。アクセス可能なツリーは、XPath 式の処理中に、(もしあれば)ステートメントによって追加されたノードのすべてのインスタンス"augment"を除去することによって、一時的に変更される。
- "when"ステートメントが"uses"、"choice"、または"case"ステートメントの子である場合、コンテキストノードは、データノードでもある"when"ステートメントを持つノードに最も近い祖先ノードになる。このようなノードが存在しない場合、コンテキストノードはルートノードになる。アクセス可能なツリーは、XPath 式の処理中に、(もしあれば)、"choice"、または"case"ステートメントによって追加されたノードのすべてのインスタンス"uses"を削除することによって、一時的に変更される。
- "when"ステートメントが他のデータ定義ステートメントの子である場合、"when"ステートメントが定義されているデータノードのすべてのインスタンスを、値と子を持たない同じ名前の単一のダミーノードに置き換えることによって、XPath 式の処理中にアクセス可能なツリーが一時的に変更される。このようなインスタンスが存在しない場合は、ダミーノードが一時的に作成される。コンテキストノードはこのダミーノードである。

XPath 式の結果は、標準 XPath 規則を使用して bool 値に変換される。

XPath 式が関連する"when"ステートメントを持つノードを参照する場合、これらの"when"式を最初に評価する必要がある(MUST)。

"when"式の間に循環従属があってはならない(MUST NOT)。

XPath 式は概念的に評価されることに注意すること。これは、実装がサーバで XPath エバリュエーターを使用する必要がないことを意味する。"when"ステートメントは、特別に記述されたコードで非常によく実装できる。

8. 制約

8.1. データの制約

いくつかの YANG ステートメントは、有効なデータに対する制約を定義する。これらの制約は、ステートメントが定義するデータ型に応じて、さまざまな方法で適用される(MUST)。

- 制約がコンフィグレーションデータに定義されている場合、有効なコンフィグレーションデータツリーでは真でなければならない(MUST)。
- 制約が状態データに定義されている場合、有効な状態データツリーにおいて真でなければならない(MUST)。
- もし制約が通知データに定義されるなら、それはどの通知データツリーでも真でなければならない(MUST)。
- 制約が RPC またはアクション入力パラメータで定義されている場合、RPC またはアクション操作の呼び出しでは真でなければならない(MUST)。
- 制約が RPC やアクションの出力パラメータで定義されている場合、RPC やアクションの応答では真でなければならない(MUST)。

次のプロパティは、すべてのデータツリーで有効である。

- すべてのリーフデータ値は、型の"range"、"length"、および"pattern"プロパティで定義されたものを含め、リーフの型制約と一致しなければならない(MUST)。
- すべてのキーリーフは、すべてのリストエントリに存在しなければならない(MUST)。
- ノードは、すべての choice の中で、多くとも 1 つの case 分岐に存在しなければなりません(MUST)。
- "if-feature"式がサーバで"false"と評価される場合、"if-feature"タグが付けられたノードが存在してはならない(MUST)。
- "when"条件がデータツリーで"false"と評価される場合、"when"タグが付けられたノードが存在してはならない(MUST)。

有効なデータツリーでは、次のプロパティが有効である。

- すべての"must"制約は"true"と評価されなければならない(MUST)。
- "path"ステートメントによって定義されたすべての参照整合性制約が満たされなければなりません(MUST)。
- リスト上のすべての"unique"制約は満たされなければならない(MUST)。
- "mandatory"制約は、ノードまたはその祖先が"when"条件または"if-feature"式を持ち、その評価が"false"でないかぎり、リーフおよび choice に適用される。
- "min-elements"制約と"max-elements"制約は、ノードまたはその祖先が"when"条件または"if-feature"式を持ち、その評価が"false"でないかぎり、リストとリーフリストに適用される。

ランニングコンフィギュレーションデータストアは常に有効である必要がある(MUST)。

8.2. コンフィグレーションデータの変更

- リクエストが choice の下にコンフィグレーションデータノードを作成する場合、データツリー内の他の case 分岐の既存のノードはサーバによって削除される。
- ノードの"when"式が false になるようにリクエストがコンフィグレーションデータノードを変更すると、"when"式を持つデータツリー内のノードがサーバによって削除される。

8.3. NETCONF 制約強制モデル

コンフィグレーションデータについては、制約が強制されなければならない場合、次の 3 つのウィンドウがある。

- RPC ペイロードの解析中
- <edit-config>操作の処理中
- 検証中

次のセクションでは、これらの各シナリオについて説明する。

8.3.1. ペイロード解析

コンテンツが RPC ペイロードに到着するとき、それは、サーバが実装するモデルのセットによって定義される階層とコンテンツルールに従って、整形式 XML でなければならない。

- リーフデータ値が、型の"range"、"length"、および"pattern"プロパティで定義されている制約を含め、リーフの型制約と一致しない場合、サーバは、"invalid-value"の<error-tag>を<rpc-error>に返し、error-app-tag (セクション 7.5.4.2 参照)と、もしあればその制約に関連する error-message (セクション 7.5.4.1 参照)を返さなければならない(MUST)。
- リストエントリのすべてのキーが存在しない場合、サーバは"missing-element"の<error-tag>を<rpc-error>に返さなければならない(MUST)。
- もし choice に複数の case 分岐のデータが存在するなら、サーバは<rpc-error>中の"bad-element"<error-tag>で応答しなければならない(MUST)。
- もし"if-feature"でタグ付けされたノードのデータが存在し、"if-feature"式がサーバで"false"と評価されるなら、サーバは<rpc-error>で"unknown-element" <error-tag>を返さなければならない(MUST)。
- もし"when"でタグ付けされたノードのデータが存在し、"when"条件が"false"と評価されるなら、サーバは<rpc-error>に"unknown-element"<error-tag>を返さなければならない(MUST)。
- 挿入処理のために、もし属性"before"と"after"の値が適切なキーリーフのタイプに対して有効でないなら、サーバは<rpc-error>の中の"bad-attribute"<error-tag>で返答しなければならない(MUST)。
- もし"ordered-by"プロパティが"user"であるリストではない要素に属性"before"と"after"が現れるなら、サーバは<rpc-error>中の"unknown-attribute" <error-tag>で返答しなければならない(MUST)。

8.3.2. NETCONF<edit-config>処理

着信データが解析された後、NETCONF サーバはデータをコンフィグレーションデータストアに適用して<edit-config>操作を実行する。この処理の間、以下のエラーを検出しなければならない(MUST)。

- 存在しないデータの削除要求。
- 既存データに対する生成要求。
- 存在しない"before"または"after"パラメータを持つ挿入要求。
- "when"でタグ付けされたノードに対する変更要求、および"when"条件は"false"と評価される。この場合、サーバは"unknown-element"の<error-tag>を<rpc-error>に含めて応答しなければならない(MUST)。

8.3.3. 検証

データストア処理が完了すると、最終的な内容はすべての検証制約に従わなければならない(MUST)。この検証処理は、データストアによって異なる時間に実行される。もしデータストアが"running"か"startup"なら、これらの制約は<edit-config>か<copy-config>操作の最後に強制されなければな

りません(MUST)。データストアが"candidate"の場合、制約の適用は、<commit>または<validate>操作が行われるまで遅延される。

9. 組み込み型

YANG には、多くのプログラミング言語と同様の組み込み型のセットがあるが、管理情報モデルの特殊な要件により、いくつかの違いがある。

これらの組み込み型または他の派生型から派生する追加型を定義できる。派生型では、サブタイプを使用して、使用可能な値のセットを正式に制限できる。

異なる組み込み型とその派生型では、異なる種類のサブタイピングが可能である。具体的には、文字列の長さと正規表現の制限(セクション 9.4.4 および 9.4.5 参照)、および数値型の範囲の制限(セクション 9.2.4 参照)である。

特定の型の値の字句表現は、XML エンコーディングと YANG モジュールでデフォルト値と数値範囲を指定するときに使用される。

9.1. 正規表現

ほとんどの型では、型の値を 1 つの正規表現で表す。同じ値の複数の字句表現を許可する型もある。例えば、正の整数"17"は、"+17"または"17"として表すことができる。　実装は、この文書で指定されたすべての字句表現をサポートしなければならない(MUST)。

サーバが XML で符号化されたデータを送信するとき、このセクションで定義された正規形を使用しなければならない(MUST)。他の符号化は代替表現を導入するかもしれない。ただし、データツリー内の値は、このセクションで定義されている正規表現に概念的に格納されることに注意すること。特に、データ型が正規形を持つ場合、XPath 式の評価は正規形を使用して行われる。データ型が正規形を持たない場合、値のフォーマットはデータ型の字句表現と一致しなければならないが、正確なフォーマットは実装に依存する(MUST)。

一部の型は、エンコーディングに依存する字句表現を持っている(例えば、それらが出現する XML コンテキスト) 。これらの型には正規形はない。

9.2. 整数組み込み型

整数組み込み型には、int8、int16、int32、int64、uint8、uint16、uint32、および uint64 がある。これらは異なるサイズの符号付き整数と符号なし整数を表す。

int8 は、-128 から 127 までの整数値を表す。

int16 は、-32768 から 32767 までの整数値を表す。

int32 は、-2147483648 から 2147483647 の整数値を表す。

int64 は、-9223372036854775808 から 9223372036854775807 までの整数値を表す。

uint8 は、0 から 255 までの整数値を表す。

uint16 は、0 から 65535 までの整数値を表す。

uint32 は、0 から 4294967295 の整数値を表す。

uint64 は、0 から 18446744073709551615 までの整数値を表す。

9.2.1. 字句表現

整数値は、字句的にはオプションの符号("+"または"-")とそれに続く一連の 10 進数として表される。符号が指定されていない場合は、"+"とみなされる。

便宜上、YANG モジュールで整数のデフォルト値を指定する場合、16 進または 8 進表記で値を表す代替字句表現を使用できる。16 進表記は、オプションの符号("+"または"-")、文字"0x"、および大文字または小文字の 16 進数字で構成される。8 進数表記は、オプションの符号("+"または"-")、文字"0"、8 進数の数字で構成される。

YANG モジュールのデフォルト値の先頭がゼロ ("0") の場合は、8 進数と解釈されることに注意すること。XML エンコーディングでは、整数は常に 10 進数として解釈され、先頭にゼロを付けることができる。

例:

```
// legal values
+4711                        // legal positive value
4711                         // legal positive value
-123                         // legal negative value
0xf00f                       // legal positive hexadecimal value
-0xf                         // legal negative hexadecimal value
052                          // legal positive octal value

// illegal values
- 1                          // illegal intermediate space
```

9.2.2. 正規形

正の整数の正規形には、符号"+"は含まれない。先行ゼロは禁止されている。値 0 は"0"として表される。

9.2.3. 制限事項

すべての整数型は、"range"ステートメントで制限できる(セクション 9.2.4 参照)。

9.2.4. "range"ステートメント

"range"ステートメントは、"type"ステートメントのオプションのサブステートメントで、引数として範囲式文字列を取る。整数および 10 進の組み込み型、またはそれらから派生した型を制限するために使用する。

範囲は、明示的な値、または下限、連続する二つのドット".."、および上限で構成される。"|"で区切られた複数の値または範囲を指定できる。複数の値または範囲が与えられる場合、それらはすべて互いに素でなければならず、昇順でなければならない(MUST)。すでに範囲制限されている型に範囲制限が適用される場合、新しい制限は等しく制限されているか、より制限されていなければなりません(MUST)。つまり、下限を上げる、上限を減らす、明示的な値または範囲を削除する、または中間ギャップを持つ複数の範囲に範囲を分割するなどである。範囲式で与えられるそれぞれの明示的な値と範囲境界値は、制限される型に一致するか、特別な値"min"か"max"のいずれかでなければなりません(MUST)。"min"および"max"は、それぞれ制限対象の型式に受け入れられる最小値および最大値を指す。

範囲式の構文は、正式にはセクション 14 の"range-arg"で定義されている。

9.2.4.1. range のサブステートメント

サブステートメント	セクション	基数
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

9.2.5. 使用例

```
typedef my-base-int32-type {
  type int32 {
    range "1..4 | 10..20";
  }
}
typedef my-type1 {
  type my-base-int32-type {
    // legal range restriction
    range "11..max"; // 11..20
  }
}
typedef my-type2 {
  type my-base-int32-type {
    // illegal range restriction
    range "11..100";
  }
}
```

9.3. decimal64 組み込み型

decimal64 組み込み型は、小数で表すことができる実数のサブセットを表す。 decimal64 の値空間は、64 ビットの符号付き整数に 10 の負の累乗を掛けることによって得られる数のセットである。つまり、"i x 10^-n"と表現できる。ここで、i は 64 ビット整数、n は 1 から 18 桁の整数である。

9.3.1. 字句表現

decimal64 値は、字句的には、オプションの符号("+"または"-")とそれに続く 10 進数のシーケンス、オプションの 10 進数インジケータ ピリオド ('.')と 10 進数のシーケンスで表す。符号が指定されていない場合は、"+"とみなされる。

9.3.2. 正規形

正の decimal64 値の正規形には、符号"+"は含まれない。小数点は必須である。小数点の前後に少なくとも 1 桁がなければならないという規則に従うことを条件として、先行ゼロおよび後続ゼロは禁止される。値 0 は"0.0"として表される(MUST)。

9.3.3. 制限事項

decimal64 型は、"range"ステートメントで制限できる(セクション 9.2.4 参照)。

9.3.4. "fraction-digits"ステートメント

"fraction-digits"ステートメントは"type"ステートメントのサブステートメントであり、型が"decimal64"の場合は存在しなければならない(MUST)。引数として、1 桁から 18 桁の整数を取る。decimal64 型の値の間の最小差のサイズを制御する。値の空間を"i x 10^-n"として表現できる数値に制限する。n は小数桁数の引数である。

次の表に、それぞれの小数桁数における最小値と最大値を示す。

小数桁数	最小	最大
1	-922337203685477580.8	922337203685477580.7
2	-92233720368547758.08	92233720368547758.07
3	-9223372036854775.808	9223372036854775.807
4	-922337203685477.5808	922337203685477.5807
5	-92233720368547.75808	92233720368547.75807
6	-9223372036854.775808	9223372036854.775807
7	-922337203685.4775808	922337203685.4775807
8	-92233720368.54775808	92233720368.54775807
9	-9223372036.854775808	9223372036.854775807
10	-922337203.6854775808	922337203.6854775807
11	-92233720.36854775808	92233720.36854775807
12	-9223372.036854775808	9223372.036854775807
13	-922337.2036854775808	922337.2036854775807
14	-92233.72036854775808	92233.72036854775807
15	-9223.372036854775808	9223.372036854775807
16	-922.3372036854775808	922.3372036854775807
17	-92.23372036854775808	92.23372036854775807
18	-9.223372036854775808	9.223372036854775807

9.3.5. 使用例

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

9.4. 組み込み型文字列

組み込み型文字列は、YANG で人間が読める文字列を表す。有効な文字は、Unicode および ISO/IEC 10646 [ISO.10646] 文字で、タブ、キャリッジ・リターンおよび改行を含むが、その他の C0 制御文字、サロゲート・ブロックおよび非文字は除外される。文字列構文は、正式にはセクション 14 の"YANG ABNF 文法"で定義されている。

9.4.1. 語彙表現

文字列値は、XML エンコーディングでは文字データとして表現される。

9.4.2. 標準形式

標準形式は語彙表現と同じである。文字列値の Unicode 正規化は実行されない。

9.4.3. 制限事項

文字列は、"length" (セクション 9.4.4)および"pattern" (セクション 9.4.5)ステートメントで制限される。

9.4.4. "length"ステートメント

"length"ステートメントは、"type"ステートメントのオプションのサブステートメントであり、長さを表現する文字列を引数として取る。組み込み型"string"と"binary"、またはこれらから派生した型を制限するために使用する。

"length"ステートメントは、文字列内の Unicode 文字の数を制限する。

長さの範囲は、明示的な値、つまり下限、連続するドット".."、および上限で構成される。 "|"で区切られた複数の値または範囲を指定できる。長さ制限値は負であってはならない(MUST NOT)。複数の値または範囲が与えられる場合、それらはすべてバラバラでなければならず、昇順でなければならない(MUST)。すでに長さ制限されている型に長さ制限が適用される場合、新しい制限は、同等かそれ以上の制限でなければならない(MUST)。すなわち、下限を上げる、上限を減らす、明示的な長さの値または範囲を削除する、または、中間ギャップを持つ複数の範囲に範囲を分割する。長さの値は、負でない整数か、特殊な値"min"または"max"のいずれかである。"min"および"max"とは、それぞれ制限対象の型式に対して許容される最小および最大長を指す。18446744073709551615 を超える長さの値をサポートするための実装は必要ない。

長さ表現の構文は、正式にはセクション 14 の規則"length-arg"で定義されている。

9.4.4.1. length のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

9.4.5. "pattern"ステートメント

"pattern"ステートメントは、"type"ステートメントのオプションのサブステートメントであり、[XSD-TYPES] で定義されているように、正規表現文字列を引数として取る。 組み込み型"string"、または"string"から派生した型を、パターンに一致する値に制限するために使用する。

型が複数の"pattern"ステートメントを持つ場合、式は AND で結合される。つまり、このような式はすべて一致する必要がある。

すでにパターン制限されている型にパターン制限を適用する場合、値は、新しいパターンに加えて、基本型のすべてのパターンに一致する必要がある。

9.4.5.1. pattern のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
modifier	9.4.6	0..1
reference	7.21.4	0..1

9.4.6. "modifier"ステートメント

"modifier"ステートメントは、"pattern"ステートメントのオプションのサブステートメントであり、文字列"invert-match"を引数として取る。

パターンに"invert-match"修飾子がある場合、型はパターンに一致しない値に制限される。

9.4.7. 使用例

次の typedef を使用する。

```
typedef my-base-str-type {
  type string {
    length "1..255";
  }
}
```

次の改良は有効である。

```
type my-base-str-type {
  // legal length refinement
  length "11 | 42..max"; // 11 | 42..255
}
```

次の改良は不正である:

```
type my-base-str-type {
  // illegal length refinement
  length "1..999";
}
```

次のタイプの場合:

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

次の文字列が一致する。

```
AB          // legal
9A00        // legal
```

次の文字列が一致しない:

```
00ABAB      // illegal, too long
xx00        // illegal, bad characters
```

次のタイプの場合:

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

次の文字列が一致する。

```
enabled     // legal
```

次の文字列が一致しない:

```
10-mbit     // illegal, starts with a number
xml-element // illegal, starts with illegal sequence
```

9.5. boolean の組み込み型

boolean 組み込み型は boolean 値を表す。

9.5.1. 語彙表現

boolean 値の語彙表現は、値が"true"または"false"の文字列である。これらの値は小文字でなければならない(MUST)。

9.5.2. 標準形式

標準形式は語彙表現と同じである。

9.5.3. 制限事項

boolean 値は制限できない。

9.6. 列挙の組み込み型

列挙組み込み型は、割り当てられた名前のセットの値を表す。

9.6.1. 語彙表現

列挙値の語彙表現は、割り当てられた名前文字列である。

9.6.2. 標準形式

標準形式は、割り当てられた名前文字列である。

9.6.3. 制限事項

列挙は、基本型の値のサブセットを列挙する"enum"ステートメント(セクション 9.6.4)を使用して制限できる。

9.6.4. "enum"ステートメント

"enum"ステートメントは"type"ステートメントのサブステートメントであり、タイプが"enumeration"の場合は存在しなければならない(MUST)。列挙型に割り当てられたそれぞれの名前を指定するために繰り返し使用される。割り当てられた名前の文字列を引数として取る。文字列は長さがゼロであってはならず(MUST NOT)、前後に空白文字("空白"プロパティを持つ任意の Unicode 文字)を含んではならない(MUST NOT)。　ユニコード制御コードの使用は避けるべきである(SHOULD)。

ステートメントの後には、オプションで、詳細な列挙情報を保持するサブステートメントのブロックが続く。　列挙で割り当てられたすべての名前は一意でなければならない(MUST)。

既存の列挙型が制限されている場合、新しい型で割り当てられた名前の集合は、基本型の割り当てられた名前の集合のサブセットでなければならない(MUST)。このような割り当てられた名前の値は変更してはならない(MUST NOT)。

9.6.4.1.　enum のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
value	9.6.4.2	0..1

9.6.4.2. "value"ステートメント

"value"ステートメントは、割り当てられた列挙名に整数値を関連付けるために使用される。この整数値は-2147483648 から 2147483647 の範囲でなければならず(MUST)、列挙型内で一意でなければならない(MUST)。

値を指定しない場合は、値が自動的に割り当てられる。"enum"サブステートメントが最初に定義された文である場合、割り当てられる値はゼロ (0) である。それ以外の場合、割り当てられた値は現在の最大 enum 値(つまり、親"enum"ステートメントの現在の"type"サブステートメントの前の暗黙的または明示的な最大 enum 値)よりも大きくなる。

"if-feature"ステートメント内に"enum"ステートメントが存在しても、自動的に割り当てられた値には影響しない。

現在の最大値が 2147483647 に等しい場合、現在の最大値に続く"enum"サブステートメントには enum 値を指定しなければなりません(MUST)。

既存の列挙型が制限されている場合、"value"ステートメントは、基本型と同じ値を持つか、または存在しない必要がある(MUST)。その場合、値は基本型と同じである。

9.6.5. 使用例

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

リーフ"myenum"の値"seven"の字句表現は、次のとおりである。

```
<myenum>seven</myenum>
```

次の typedef を使用する。

```
typedef my-base-enumeration-type {
  type enumeration {
    enum white {
      value 1;
    }
    enum yellow {
      value 2;
    }
    enum red {
      value 3;
    }
  }
"
```

次の改良は有効である。

```
type my-base-enumeration-type {
  // legal enum refinement
  enum yellow;
  enum red {
    value 3;
  }
}
```

次の改良は不正である:

```
type my-base-enumeration-type {
  // illegal enum refinement
  enum yellow {
    value 4; // illegal value change
  }
  enum black; // illegal addition of new name
}
```

次の例では、"enum"に"if-feature"というタグを付けて、対応する機能をアドバタイズするサーバだけで値を有効にする方法を示す。

```
type enumeration {
  enum tcp;
  enum ssh {
    if-feature ssh;
  }
  enum tls {
  if-feature tls;
  }
}
```

9.7. ビット組み込み型

ビット組み込み型はビットセットを表す。　すなわち、ビット値は、0 から始まる小さな整数位置番号によって識別されるフラグのセットである。　各ビット番号には名前が割り当てられている。

既存のビット型が制限されている場合、新しい型の割り当てられた名前の集合は、基本型の割り当てられた名前の集合のサブセットでなければならない(MUST)。そのような割り当てられた名前のビット位置は変更されてはならない(MUST)。

9.7.1. 制限事項

ビットタイプは、"bit"ステートメント(セクション 9.7.4)で制限できる。

9.7.2. 語彙表現

bits type の語彙表現は、設定されるビットの名前をスペースで区切ったリストである。 したがって、長さ 0 の文字列は、ビットが設定されていない値を表す。

9.7.3. 標準形式

標準的な形式では、ビット値は単一のスペース文字で区切られ、位置順に表示される (セクション 9.7.4.2 を参照。)。

9.7.4. "bit"ステートメント

"bit"ステートメントは"type"ステートメントのサブステートメントであり、タイプが"bit"の場合は存在しなければならない (MUST)。ビットタイプの名前付きビットを指定するために繰り返し使用される。この関数は、ビットに割り当てられた名前の文字列を引数として取る。その後に、詳細なビット情報を保持するサブステートメントのブロックが続く。割り当てられた名前は、識別子 (セクション 6.2 参照) と同じ構文規則に従う。 ビットタイプで割り当てられたすべての名前は一意でなければならない (MUST)。

9.7.4.1. bit のサブステートメント

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
position	9.7.4.2	0..1
reference	7.21.4	0..1
status	7.21.2	0..1

9.7.4.2. "position"ステートメント

"position"ステートメントはオプションで、仮想ビットフィールド内のビットの位置を指定する非負の整数値を引数として取る。位置値は 0 から 4294967295 の範囲でなければならず、ビットタイプ内で一意でなければならない (MUST)。

ビット位置が指定されていない場合は、自動的に割り当てられる。"bit"サブステートメントが最初に定義された文である場合、割り当てられる値はゼロ (0) である。そうでなければ、割り当てられた値は現在の最高ビット位置 (すなわち、親"bit"ステートメントの現在の"type"サブステートメントの前の暗黙的または明示的な最上位ビット位置) よりも一つ大きい。

"if-feature"ステートメントが"bit"ステートメントに存在しても、自動的に割り当てられた位置には影響しない。

現在の最上位ビット位置値が 4294967295 に等しい場合、現在の最上位ビット位置値に続く"bit"サブステートメントに対して位置値を指定しなければならない (MUST)。

既存のビットタイプが制限されている場合、"position"ステートメントは、基本タイプと同じ値を持つか、または存在しない必要がある (MUST)。その場合、値は基本タイプと同じである。

9.7.5. 使用例

次の typedef と leaf を指定する。

```
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";
}
```

ビット値が disable-nagle のこのリーフの語彙表現と ten-mb-only set は以下のようになる。

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

次の例は、この型の正規表現を示している。

```
type mybits-type {
  // legal bit refinement
  bit disable-nagle {
    position 0;
  }
  bit auto-sense-speed {
    position 1;
  }
}
```

次の改良は不正である:

```
type mybits-type {
  // illegal bit refinement
  bit disable-nagle {
    position 2; // illegal position change
  }
  bit hundred-mb-only; // illegal addition of new name
}
```

9.8. binary 組み込み型

binary 組み込み型は、任意の binary データ、つまりオクテットのシーケンスを表す。

9.8.1. 制限事項

binary 型は、"length" (セクション 9.4.4) ステートメントで制限できる。binary 値の長さは、含まれるオクテット数である。

9.8.2. 字句表現

binary 値は base 64 符号化方式で符号化される ([RFC 4648] のセクション 4).

9.8.3. 正規型

binary 値の正規型は、"Base 64 Encoding"[RFC 4648]の規則に従う。

9.9. leafref 組み込み型

leafref 組み込み型は、スキーマツリー内のリーフノードまたはリーフリストノードの値空間に制限され、オプションでデータツリー内の対応するインスタンスノードによってさらに制限される。"path"サブステートメント (セクション 9.9.2) は、スキーマ・ツリー内の参照されるリーフ・ノードまたはリーフ・リスト・ノードを識別するために使用される。参照ノードの値空間は参照ノードの値空間である。

"require-instance"プロパティ (セクション 9.9.3) が"true"の場合、参照されるスキーマツリーリーフまたはリーフリストノードのデータツリー内に、有効なデータツリー内の leafref 値と同じ値を持つノード、または使用中のデフォルト値(セクション 7.6.1 および 7.7.2 参照)を持つノードが存在しなければならない(MUST)。

参照元ノードが設定データを表し、"require-instance"プロパティ (セクション 9.9.3) が"true"の場合、参照先ノードは設定を再度表していなければならない(MUST)。leafref の円形チェーンがあってはならない(MUST NOT)。

leafref が参照するリーフが一つ以上の特徴 (セクション 7.20.2 を参照) に基づいて条件付きである場合、leafref 型のリーフも少なくとも同じ特徴の集合に基づいて条件付きでなければならない(MUST)。

9.9.1. 制限事項

leafref は、"require-instance"ステートメントで制限できる (セクション 9.9.3)。

9.9.2. "path"ステートメント

"path"ステートメントは"type"ステートメントのサブステートメントであり、タイプが"leafref"の場合は存在しなければならない(MUST)。引数として、リーフノードまたはリーフリストノードを参照しなければならない文字列を取る(MUST)。

path 引数の構文は、XPath 省略構文のサブセットである。述語は、リストエントリのキーノードの値を制約する場合にのみ使用される。各述語は、キーごとにちょうど 1 つの等価性テストから構成され、リストが複数のキーを持つ場合、複数の隣接する述語が存在してもよい(MAY)。この構文は、正式にはセクション 14 の規則"path-arg"で定義されている。

述語は、リーフインスタンスを一意に識別するために複数のキー参照が必要な場合にのみ使用される。これは、リストに複数のキーがある場合や、リスト内のキー以外のリーフへの参照が必要な場合に発生する。このような場合、通常は複数のリーフ参照が指定され、それらを結び付けるために述部が使用される。

"path"式は、0、1 つまたは複数のノードで構成されるノードセットに評価される。 "require-instance"プロパティが"true"の場合、このノードセットは空であってはならない。

"path" XPath 式は、セクション 6.4.1 の定義に加えて、次のコンテキストで概念的に評価される。

- "path"ステートメントが typedef 内で定義されている場合、コンテキストノードは typedef を参照するデータツリー内のリーフノードまたはリーフリストノードである。
- それ以外の場合、コンテキストノードは"path"ステートメントが定義されているデータツリー内のノードである。

9.9.3. "require-instance"ステートメント

"require-instance"ステートメントは、"type"ステートメントのサブステートメントであり、タイプが"instance-identifier"または"leafref"の場合に存在してもよい(MAY)。引数として文字列"true"または"false"を取る。このステートメントが存在しない場合は、デフォルトの"true"が使用される。

もし"require-instance"が"true"なら、それは参照されているインスタンスが、データが有効であるために存在しなければならないことを意味する MUST)。この制約はセクション 8 の規則に従って強制される。

"require-instance"が"false"の場合、参照されているインスタンスが有効なデータ内に存在してもよいことを意味する(MAY)。

9.9.4. 字句表現

leafref 値は、参照するリーフがその値を表すのと同じように、字句的に表現される。

9.9.5. 正規形

leafref の正規形は、参照するリーフの正規形と同じである。

9.9.6. 使用例

次のリストを使用する

```
list interface {
  key "name";
  leaf name {
    type string;
  }
  leaf admin-status {
    type admin-status;
  }
  list address {
    key "ip";
    leaf ip {
      type yang:ip-address;
    }
```

```
    }
  }
```

次の leafref は既存のインターフェイスを参照する

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

対応する XML の断片の例:

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

次のリーフリファレンスは、インターフェイスの既存のアドレスを参照する

```
  container default-address {
    leaf ifname {
      type leafref {
        path "../../interface/name";
      }
    }
    leaf address {
      type leafref {
        path "../../interface[name = current()/../ifname]"
           + "/address/ip";
      }
    }
  }
```

対応する XML の断片の例:

```
  <interface>
    <name>eth0</name>
    <admin-status>up</admin-status>
```

```
    <address>
      <ip>192.0.2.1</ip>
    </address>
    <address>
      <ip>192.0.2.2</ip>
    </address>
  </interface>
  <interface>
    <name>lo</name>
    <admin-status>up</admin-status>
    <address>
      <ip>127.0.0.1</ip>
    </address>
  </interface>
  <default-address>
    <ifname>eth0</ifname>
    <address>192.0.2.2</address>
  </default-address>
```

次のリストでは、キーの 1 つに leafref を使用している。これは、リレーショナルデータベースの外部キーに似ている。

```
list packet-filter {
  key "if-name filter-id";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf filter-id {
    type uint32;
  }
  ...
}
```

対応する XML の断片の例:

```
<interface>
  <name>eth0</name>
  <admin-status>up</admin-status>
  <address>
    <ip>192.0.2.1</ip>
  </address>
```

```
    <address>
      <ip>192.0.2.2</ip>
    </address>
  </interface>
  <packet-filter>
    <if-name>eth0</if-name>
    <filter-id>1</filter-id>        ...
  </packet-filter>
  <packet-filter>
    <if-name>eth0</if-name>
    <filter-id>2</filter-id>        ...
  </packet-filter>
```

以下の通知では、既存の管理ステータス:

```
  notification link-failure {
    leaf if-name {
      type leafref {
        path "/interface/name";
      }
    }
    leaf admin-status {
      type leafref {
        path "/interface[name = current()/../if-name]"
           + "/admin-status";
      }
    }
  }
```

対応する XML 通知の例を次に示す。

```
  <notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2008-04-01T00:01:00Z</eventTime>
    <link-failure xmlns="urn:example:system">
      <if-name>eth0</if-name>
      <admin-status>up</admin-status>
    </link-failure>
  </notification>
```

9.10. identityref 組み込み型

identityref 組み込み型は、既存の ID (セクション 7.18 参照)を参照するために使用される。

9.10.1. 制限事項

identityref は制限できません。

9.10.2. identityref の"base"ステートメント

"base"ステートメントは"type"ステートメントのサブステートメントであり、型が"identityref"の場合は少なくとも一度は存在しなければならない(MUST)。引数は、"identity"ステートメントで定義される ID の名前である。識別名にプレフィクスが存在する場合、そのプレフィクスを使用してインポートされたモジュールで定義されている識別を示す。そうでなければ、名前が一致する ID を現在のモジュールまたは含まれるサブモジュールで定義しなければならない(MUST)。

identityref の有効な値は、すべての identityref の基本 ID から派生した任意の ID である。特定のサーバでは、有効な値は、サーバによって実装されたモジュールで定義された ID のセットにさらに制限される。

9.10.3. 字句表現

identityref は、 [XML-NAMES] で定義されているように、参照される ID の修飾名として字句的に表現される。接頭辞がない場合、identityref の名前空間は、identityref 値を含む要素で有効なデフォルトの名前空間になる。

identityref に"default"ステートメントを使用してデフォルト値を指定する場合、デフォルト値内の ID 名に接頭辞を付けてもよい。識別名にプレフィクスが存在する場合は、そのプレフィクスを使用してインポートされたモジュールで定義された ID を参照する。現在のモジュールまたはそのサブモジュールの 1 つで ID が定義されている場合は、現在のモジュールのプレフィクスを参照する。そうでない場合、名前が一致する ID は、現在のモジュールまたはそのサブモジュールの 1 つで定義されなければならない(MUST)。

"must"または"when" XPath 式の"identityref"のノードの文字列値は、接頭辞が存在する参照先 ID の修飾名である。参照される ID がインポートされたモジュールで定義されている場合、文字列値のプレフィクスは、対応する"import"ステートメントで定義されているプレフィクスである。それ以外の場合、文字列値のプレフィクスは現在のモジュールのプレフィクスである。

9.10.4. 正規形

字句形式は値が出現する XML コンテキストに依存するため、この型は正規形を持ちません。

9.10.5. 使用例

セクション 7.18.3 の ID 定義および次のモジュールを使用する。

```
module example-my-crypto {
  yang-version 1.1;
  namespace "urn:example:my-crypto";
  prefix mc;
  import "example-crypto-base" {
    prefix "crypto";
  }
  identity aes {
    base "crypto:crypto-alg";
  }
  leaf crypto {
```

```
        type identityref {
          base "crypto:crypto-alg";
        }
      }
      container aes-parameters {
        when "../crypto = 'mc:aes'";
        ...
      }
    }
```

値が"des"モジュールで定義された"des3" identity である場合、"crypto"リーフがどのようにエンコードされるかの例を次に示す。

```
<crypto xmlns:des="urn:example:des">des:des3</crypto>
```

エンコーディングで使用されるプレフィックスは、各インスタンスエンコーディングに対してローカルである。これは、同一の identityref が異なる実装によって異なるように符号化される可能性があることを意味する。たとえば、次の例では、上記と同じリーフをエンコードする。

```
<crypto xmlns:des="urn:example:des">des:des3</crypto>
```

"crypto"リーフの値が"aes"モジュールで定義されている"example-my-crypto"の場合、次のようにエンコードできる。

```
<crypto xmlns:mc="urn:example:my-crypto">mc:aes</crypto>
```

または、デフォルトのネームスペースを使用する。

```
  <crypto>aes</crypto>
```

9.11. empty の組み込み型

empty の組み込み型は、値を持たないリーフを表す。その有無によって情報を伝達する。 empty の型にはデフォルト値を設定できない。

9.11.1. 制限事項

empty のタイプは制限できない。

9.11.2. 字句表現

該当なし。

9.11.3. 正規形

該当なし。

9.11.4. 使用例

次のリーフの場合:

```
leaf enable-qos {
  type empty;
}
```

リーフが存在する場合の有効なエンコーディングの例を次に示す。

```
<enable-qos/>
```

9.12. union 組み込み型

union 組み込み型は、その member 型の 1 つに対応する値を表す。

型が union の場合、"type"ステートメント(セクション 7.4)が存在しなければならない(MUST)。union の各 member 型を指定するために繰り返し使用される。これは、member 型の名前である文字列を引数として取る。member 型には、任意の組み込み型または派生型を使用できる。

XML エンコードを生成する場合、値は、その値が属する member 型の規則に従ってエンコードされる。XML エンコーディングを解釈する場合、値は、一致が見つかるまで、"type"ステートメントで指定された順序で、各 member 型に対して連続して検証される。　一致した型は検証されたノードの値の型となり、エンコーディングはその型の規則に従って解釈される。

member 型で定義された既定値または"units"プロパティは、union 型に継承されない。

9.12.1. 制限事項

union は制限できません。ただし、各 member 型は、セクション 9 で定義されている規則に基づいて制限できる。

9.12.2. 字句表現

union の字句表現は、いずれかのメンバー型の表現に対応する値である。

9.12.3. 正規形

union 値の正規形は、値の member 型の正規形と同じである。

9.12.4. 使用例

次に、int 32 と enumeration を示す。

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

member 型が leafref で、"require-instance"プロパティ(セクション 9.9.3)が"true"の場合は注意が必要である。このような型のリーフが既存のインスタンスを参照している場合、ターゲットインスタンスが削除されると、リーフの値を再検証する必要がある。　たとえば、次のように定義する

```
list filter {
  key name;
  leaf name {
    type string;
  }
  ...
}
leaf outbound-filter {
  type union {
    type leafref {
      path "/filter/name";
    }
    type enumeration {
      enum default-filter;
    }
  }
}
```

フィルタリストに"http"という名前のエントリが存在し、outbound-filter リーフにこの値があるとする。

```
<filter>
  <name>http</name>
</filter>
<outbound-filter>http</outbound-filter>
```

フィルタエントリ"http"を削除すると、outbound-filter リーフの値が leafref と一致しなくなり、次の member 型がチェックされる。現在の値 ("http")は列挙と一致しないため、結果の構成は無効である。

union の二番目の member 型が列挙型ではなく"string"型であった場合、現在の値が一致し、結果の構成が有効になる。

9.13. instance-identifier 組み込み型

instance-identifier 組み込み型は、データツリー内の特定のインスタンスノードを一意に識別するために使用される。

instance-identifier の構文は、XPath 省略構文のサブセットで、正式にはセクション 14 の規則"instance-identifier"で定義されている。　データツリー内のノードを一意に識別するために使用される。　述語は、リストエントリのキーノードの値、リーフリストエントリの値、またはキーのないリストの位置インデックスを指定する場合にのみ使用される。　キーを持つリストエントリを識別するために、各述語はキーごとに 1 つの等価性テストで構成され、各キーは対応する述語を持たなければならない(MUST)。　キーの種類が"empty"の場合、長さゼロの文字列 ("") として表される。

instance-identifier 型のリーフが設定データを表し、"require-instance"プロパティ(セクション 9.9.3 参照)が"true"の場合、参照するノードは設定も表さなければならない(MUST)。　このようなリーフは、有効なデータに制約を課す。　そのようなリーフノードはすべて、データを有効にするために、使用中のデフォルト値(セクション 7.6.1 およびセクション 7.7.2 参照)で既存のノードまたはリーフあるいはリーフリストノードを参照しなければならない(MUST)。　この制約はセクション 8 の規則に従って強制される。

"instance-identifier" XPath 式は、セクション 6.4.1 の定義に加えて、次のコンテキストで概念的に評価される。

o コンテキストノードは、アクセス可能なツリーのルートノードである。

9.13.1. 制限事項

instance-identifier は、"require-instance"ステートメントを使用して制限できる(セクション 9.9.3 参照)。

9.13.2. 字句表現

instance-identifier の値は、字句的には文字列として表される。　instance-identifier の値に含まれるすべてのノード名は、明示的な名前空間プレフィクスで修飾する必要がある。また、これらのプレフィクスは、instance-identifier の XML 要素の XML 名前空間スコープで宣言する必要がある。

エンコーディングで使用されるプレフィクスは、各インスタンスエンコーディングに対してローカルである。　これは、同じ instance-identifier が異なる実装によって異なるように符号化されるかもしれないことを意味する。

9.13.3. 正準形式

字句形式は値が出現する XML コンテキストに依存するため、この型は正規形式を持たない。

9.13.4. 使用例

instance-identifier の例を次に示す。

```
/* instance-identifier for a container */
 /ex:system/ex:services/ex:ssh

 /* instance-identifier for a leaf */
 /ex:system/ex:services/ex:ssh/ex:port

 /* instance-identifier for a list entry */
 /ex:system/ex:user[ex:name='fred']

 /* instance-identifier for a leaf in a list entry */
 /ex:system/ex:user[ex:name='fred']/ex:type

 /* instance-identifier for a list entry with two keys */
 /ex:system/ex:server[ex:ip='192.0.2.1'][ex:port='80']
```

```
/* instance-identifier for a list entry where the second
   key ("enabled") is of type "empty" */
/ex:system/ex:service[ex:name='foo'][ex:enabled='']

/* instance-identifier for a leaf-list entry */
/ex:system/ex:services/ex:ssh/ex:cipher[.='blowfish-cbc']

/* instance-identifier for a list entry without keys */
/ex:stats/ex:port[3]
```

10. XPath 関数

この文書では、2 つの汎用 XPath 関数と 5 つの YANG 型固有 XPath 関数を定義する。 関数シグネチャは、 [XPATH] で使用される構文で指定される。

10.1. ノードセットの関数

10.1.1. current()

```
node-set current()
```

current()関数は、入力パラメータを取らず、初期コンテキストノードを唯一のメンバとして持つノードセットを返す。

10.1.1.1. 使用例

次のリストを使用する。

```
list interface {
  key "name";
  ...
  leaf enabled {
    type boolean;
  }
  ...
}
```

次のリーフは、参照されるインターフェイスが有効であることを保証する"must"式を定義する。

```
leaf outgoing-interface {
  type leafref {
    path "/interface/name";
  }
  must '/interface[name=current()]/enabled = "true"';
}
```

10.2. 文字列の関数

10.2.1. re-match()

```
boolean re-match(string subject, string pattern)
```

re-match()関数は"subject"文字列が"pattern"の正規表現と一致した場合に"true"を返す。それ以外の場合は"false"を返す。

re-match()関数は、文字列が指定した正規表現と一致するかどうかを調べる。 使用される正規表現は、XML Schema の正規表現 [XSD-TYPES] である。 これには、正規表現の先頭と末尾への暗黙的なアンカーが含まれることに注意すること。

10.2.1.1. 使用例

式:

```
re-match("1.22.333", "\d{1,3}\.\d{1,3}\.\d{1,3}")
```

"true"が返される。

eth 0.<number>と呼ばれるすべての論理インターフェイスをカウントするには、次のようにする。

```
count(/interface[re-match(name, "eth0\.\d+")])
```

10.3. YANG 型の関数"leafref"および"instance-identifier"

10.3.1. deref()

```
node-set deref(node-set nodes)
```

deref()関数は、引数"nodes"のドキュメント順で最初のノードによって定義された参照の後に続き、参照されたノードを返す。

最初の引数ノードの型が"instance-identifier"の場合、この関数は、instance identifier が参照する単一のノードが存在すれば、そのノードを含むノードセットを返す。 該当するノードが存在しない場合は、空のノードセットが返される。

最初の引数ノードの型が"leafref"の場合、関数は leafref が参照するノードを含むノードセットを返す。 具体的には、このセットには、leafref の"path"ステートメント(セクション 9.9.2)によって選択された、最初の引数ノードと同じ値を持つノードが含まれる。

最初の引数ノードがその他の型の場合は、空のノードセットが返される。

10.3.1.1. 使用例

```
list interface {
  key "name type";
  leaf name { ... }
  leaf type { ... }
  leaf enabled {
    type boolean;
  }
  ...
}

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

```
        path "/interface[name=current()/../name]/type";
      }
      must 'deref(.)/../enabled = "true"' {
        error-message
          "The management interface cannot be disabled.";
      }
    }
  }
```

10.4. YANG 型の関数"identityref"

10.4.1. derived-from()

```
boolean derived-from(node-set nodes, string identity)
```

derived-from()関数は、引数"nodes"のいずれかのノードが"identityref"型のノードであり、その値が引数"identity"から派生した identity である場合(セクション 7.18.2 参照)、"true"を返す。それ以外の場合は"false"を返す。

パラメータ"identity"は、セクション 14 の規則"identifier-ref"に一致する文字列である。ID にプレフィクスが存在する場合は、そのプレフィクスを使用してインポートされたモジュールで定義されている ID を指し、プレフィクスがローカルモジュールのプレフィクスと一致する場合はローカルモジュールを指す。　プレフィクスが存在しない場合、ID は現在のモジュールまたは含まれているサブモジュールで定義されている ID を参照する。

10.4.1.1.　使用例

```
module example-interface {
  yang-version 1.1;
  ...
  identity interface-type;
  identity ethernet {
    base interface-type;
  }
  identity fast-ethernet {
    base ethernet;
  }
  identity gigabit-ethernet {
    base ethernet;
  }
  list interface {
    key name;
    ...
    leaf type {
       type identityref {
         base interface-type;
       }
    }
```

```
    ...
  }
  augment "/interface" {
    when 'derived-from(type, "exif:ethernet")';
    // generic Ethernet definitions here
  }
  ...
}
```

10.4.2. derived-from-or-self()

```
boolean derived-from-or-self(node-set nodes, string identity)
```

derived-from-or-self()関数は、引数"nodes"内のいずれかのノードが"identityref"型のノードであり、その値が identity (セクション 7.18.2 参照)と等しいか、またはそこから派生した identity である場合、"true"を返す。それ以外の場合は"false"を返す。

パラメータ"identity"は、セクション 14 の規則"identifier-ref"に一致する文字列である。 ID にプレフィクスが存在する場合は、そのプレフィクスを使用してインポートされたモジュールで定義されている ID を指し、プレフィクスがローカルモジュールのプレフィクスと一致する場合はローカルモジュールを指す。 プレフィクスが存在しない場合、ID は現在のモジュールまたは含まれているサブモジュールで定義されている ID を参照する。

10.4.2.1. 使用例

セクション 10.4.1.1 で定義されているモジュールには、次のようなものもある。

```
augment "/interface" {
  when 'derived-from-or-self(type, "exif:fast-ethernet");
  // Fast-Ethernet-specific definitions here
}
```

10.5. YANG 型の関数"enumeration"

10.5.1. enum-value()

```
number enum-value(node-set nodes)
```

enum-value()関数は、引数"nodes"のドキュメント順で最初のノードが"enumeration"型のノードかどうかを調べ、enum の整数値を返す。 "nodes"ノードセットが空の場合、または"nodes"の最初のノードが型"enumeration"でない場合は、NaN (数ではない)が返される。

10.5.1.1. 使用例

このデータモデルを使用する。

```
list alarm {
  ...
  leaf severity {
    type enumeration {
```

```
        enum cleared {
          value 1;
        }
        enum indeterminate {
          value 2;
        }
        enum minor {
          value 3;
        }
        enum warning {
          value 4;
        }
        enum major {
          value 5;
        }
        enum critical {
          value 6;
        }
      }
    }
  }
```

次の XPath 式は、重大度が"major"以上のアラームだけを選択する。

```
/alarm[enum-value(severity) >= 5]
```

10.6. YANG 型の関数"bits"

10.6.1. bit-is-set()

```
boolean bit-is-set(node-set nodes, string bit-name)
```

bit-is-set()関数は、引数"nodes"のドキュメント順の最初のノードが"bits"型のノードで、その値に"bit-name"がセットされている場合、"true"を返す。それ以外の場合は"false"を返す。

10.6.1.1. 使用例

インターフェイスに次のリーフがある場合:

```
leaf flags {
  type bits {
    bit UP;
    bit PROMISCUOUS;
    bit DISABLED;
  }
}
```

次の XPath 式を使用して、UP フラグが設定されているすべてのインターフェイスを選択できる。

```
/interface[bit-is-set(flags, "UP")]
```

11. モジュールの更新

モジュールで経験を積むと、そのモジュールを改訂することが望ましい場合がある。　ただし、公開されたモジュールを変更すると、元の仕様を使用するクライアントと更新された仕様を使用するサーバとの間で相互運用性の問題が発生する可能性がある場合は、変更できない。

公開された変更については、新しい"revision"ステートメント(セクション 7.1.9)を既存の"revision"ステートメントの前に含めなければならない(MUST)。　既存の"revision"ステートメントがない場合、新しいリビジョンを識別するためにステートメントを追加する必要がある(MUST)。　さらに、必要な変更は、"organization"および"contact"ステートメントを含むすべてのメタデータステートメントに適用しなければならない(MUST)(セクション 7.1.7 および 7.1.8)。

モジュールに含まれる定義は、他のモジュールからインポートでき、モジュール名を使用して"import"ステートメントで参照される。　したがって、モジュール名を変更してはならない(MUST NOT)。　さらに、すべての XML 要素が名前空間によって修飾されるため、"namespace"ステートメントを変更してはならない(MUST NOT)。

古い定義は、発行されたモジュールから削除してはならない(MUST NOT)。なぜなら、それらの識別子は、他のモジュールから参照されている可能性があるからである。

公開されたモジュールの定義は、次のいずれかの方法で変更できる。

- 古い enum の値が変更されない限り、"enumeration"型には新しい enum が追加される。　既存の enum の前に新しい enum を挿入したり、既存の enum を並べ替えたりすると、明示的な値が割り当てられていない限り、既存の enum に新しい値が作成されることに注意すること。
- "bits"型は、古いビット位置が変わらない限り、新しいビットを追加することができる。　既存のビットの前に新しいビットを挿入したり、既存のビットを並べ替えたりすると、明示的な位置が割り当てられていない限り、既存のビットの新しい位置になる。
- "range"、"length"、または"pattern"ステートメントは、許可される値の範囲を拡張できる。
- "default"ステートメントは、デフォルト値(型を介して直接的または間接的に)を持たないリーフに追加することができる。
- "units"ステートメントを追加することができる。
- "reference"ステートメントを追加または更新できる。
- "must"ステートメントを削除するか、その制約を緩和できる。
- "when"ステートメントを削除するか、その制約を緩和できる。
- "mandatory"ステートメントは削除するか、または"true"から"false"に変更できる。
- "min-elements"ステートメントは、必要な要素が少なくなるように削除または変更できる。
- "max-elements"ステートメントを削除または変更して、より多くの要素を使用できるようにすることができる。
- "description"ステートメントは、定義の意味を変更することなく追加または変更できる。
- "base"ステートメントは、"identity"ステートメントに追加することができる。
- "base"ステートメントは、"identityref"型から削除できる。ただし、"base"ステートメントが少なくとも 1 つ残っている必要がある。
- 新しい typedef、grouping、rpc、notification、extension、feature、および identity を追加できる。
- 新しいデータ定義ステートメントは、必須ノード(セクション 3)を既存のノードに追加しない場合、モジュールまたはサブモジュールの最上位レベルに追加しない場合、または条件付きで新しい機能に依存する場合(つまり、新しい feature を参照する"if-feature"ステートメントがある場合)に追加できる。
- 新しい"case"ステートメントを追加できる。
- 状態データを表していたノードは、必須でない限り、構成を表すように変更できる(セクション 3)。

- ノードが必須でない場合は、"if-feature"ステートメントを削除できる(セクション 3)。
- "status"ステートメントは、追加、"current"から"deprecated"または"obsolete"への変更、または"deprecated"から"obsolete"への変更が可能である。
- "type"ステートメントは、型の構文または意味を変更しない別の"type"ステートメントで置き換えることができる。　たとえば、インライン型定義は typedef に置き換えることができるが、int8 型は構文が変わるため、int16 に置き換えることはできない。
- データ定義ノードの任意のセットを、構文的および意味的に等価なノードの別のセットで置き換えることができる。　たとえば、リーフのセットは、同じリーフでグループ化された"uses"ステートメントで置き換えることができる。
- モジュール内の定義がここで許可されている以外の方法で変更されない限り、モジュールをサブモジュールのセットに分割したり、サブモジュールを削除したりできる。
- "prefix"ステートメントは、プレフィクスのローカルでの使用もすべて変更される場合は、変更できる。

さもなければ、以前の定義の意味が変更された場合(すなわち、上記で特に認められた定義以外の定義に編集上の変更が加えられた場合)、これは新しい識別子を持つ新しい定義によって達成されなければならない(MUST)。

サブステートメントとしてデータ定義ステートメントを持つステートメントでは、これらのデータ定義サブステートメントを並べ替えてはならない(MUST NOT)。　新しいデータ定義ステートメントを追加する場合は、既存のサブステートメントのシーケンスの任意の場所に追加できる。

12. YANG Version 1 との共存

YANG バージョン 1.1 モジュールは YANG バージョン 1 サブモジュールを含んではならず、YANG バージョン 1 モジュールは YANG バージョン 1.1 サブモジュールを含んではならない(MUST NOT)。

YANG バージョン 1 モジュールまたはサブモジュールは、YANG バージョン 1.1 モジュールをリビジョンでインポートしてはならない(MUST NOT)。

YANG バージョン 1.1 モジュールまたはサブモジュールは、YANG バージョン 1 モジュールをリビジョンでインポートしてもよい(MAY)。

YANG バージョン 1 のモジュール A がリビジョンなしでモジュール B をインポートし、モジュール B が YANG バージョン 1.1 にアップデートされた場合、サーバはこれらのモジュール(A と B)を同時に実装してもよい(MAY)。　このような場合、NETCONF サーバは、セクション 5.6.4 で定義された規則を使用して両方のモジュールをアドバタイズしなければならず(MUST)、[RFC 6020]で定義された規則に従った YANG バージョン 1 で規定された、モジュール A と最新のリビジョンのモジュール B をアドバタイズすべきである。(SHOULD)。

この規則は、既存の YANG バージョン 1 モジュールを YANG バージョン 1.1 モジュールとともに実装できるようにするために存在する。　この規則がない場合、単一のモジュールを YANG バージョン 1.1 に更新すると、それをインポートするモジュールに連鎖的な影響があり、すべてのモジュールを YANG バージョン 1.1 に更新する必要がある。

13. YIN

YANG モジュールは、YIN と呼ばれる別の XML ベースの構文に変換できる。変換されたモジュールは YIN モジュールと呼ばれる。　ここでは、2 つの形式間の双方向マッピング規則について説明する。

YANG 形式と YIN 形式には、異なる表記を使用した同等の情報が含まれる。　YIN 表記法を使用すると、開発者は YANG データ・モデルを XML で表現できるため、データのフィルタリングと検証、コードとドキュメントの自動生成、その他のタスクに XML ベースの豊富なツールセットを使用できる。　XSLT や XML 検証ツールなどのツールを利用できる。

YANG と YIN のマッピングでは、モデルの情報コンテンツは変更されない。　コメントと空白は保持されない。

13.1. 正式な YIN 定義

YANG キーワードと YIN 要素は 1 対 1 で対応している。　YIN 要素のローカル名は、対応する YANG キーワードと同じである。　これは、特に、YIN 文書の文書要素 (ルート) が常に<module>または<submodule>であることを意味する。

YANG キーワードに対応する YIN 要素は、関連付けられた URI が "urn:ietf:params:xml:ns:yang:yin:1"である名前空間に属する。

拡張キーワードに対応する YIN 要素は、拡張キーワードが"extension"ステートメントによって宣言されている YANG モジュールの名前空間に属す。

すべての YIN 要素の名前は、(上記の通り)の標準的なメカニズム、すなわち"XML 形式"と "xmlns:xxx"属性を使用して、名前空間で適切に修飾されなければならない(MUST)。

YANG ステートメントの引数は、XML 属性またはキーワード要素のサブ要素として YIN で表される。表 1 は、YANG キーワードのセットのマッピングを定義している。　拡張の場合、引数の対応付けは "extension"ステートメントの中で指定される(セクション 7.19 参照)。　引数には次の規則が適用される。

- 引数が属性として表される場合、この属性には名前空間がない。
- 引数が要素として表される場合、親キーワード要素と同じ名前空間で修飾される。
- 引数が要素として表される場合、それはキーワード要素の最初の子でなければならない(MUST)。

YANG ステートメントのサブステートメントは、キーワード要素の (追加の) 子として表され、その相対的な順序は、YANG のサブステートメントの順序と同じである必要がある。YANG のコメントは、XML コメントにマップされる可能性がある。

keyword	argument name	yin-element
action	name	false
anydata	name	false
anyxml	name	false
argument	name	false
augment	target-node	false
base	name	false
belongs-to	module	false
bit	name	false
case	name	false
choice	name	false

config	value	false
contact	text	true
container	name	false
default	value	false
description	text	true
deviate	value	false
deviation	target-node	false
enum	name	false
error-app-tag	value	false
error-message	value	true
extension	name	false
feature	name	false
fraction-digits	value	false
grouping	name	false
identity	name	false
if-feature	name	false
import	module	false
include	module	false
input	<no argument>	n/a
key	value	false
leaf	name	false
leaf-list	name	false
length	value	false
list	name	false
mandatory	value	false
max-elements	value	false
min-elements	value	false
modifier	value	false
module	name	false
must	condition	false
namespace	uri	false
notification	name	false
ordered-by	value	false
organization	text	true
output	<no argument>	n/a
path	value	false
pattern	value	false
position	value	false
prefix	value	false
presence	value	false
range	value	false
reference	text	true
refine	target-node	false

require-instance	value	false
revision	date	false
revision-date	date	false
rpc	name	false
status	value	false
submodule	name	false
type	name	false
typedef	name	false
unique	tag	false
units	name	false
uses	name	false
value	value	false
when	condition	false
yang-version	value	false
yin-element	value	false

表 1:YANG ステートメントの引数のマッピング

13.1.1. 使用例

次の YANG モジュール:

```
module example-foo {
  yang-version 1.1;
  namespace "urn:example:foo";
  prefix "foo";
  import example-extensions {
    prefix "myext";
  }
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf mtu {
      type uint32;
      description "The MTU of the interface.";
      myext:c-define "MY_MTU";
    }
  }
}
```

ここで、拡張子"c-define 関数 c-define"はセクション 7.19.3 で定義されており、以下の YIN に変換される:

```
<module name="example-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:foo="urn:example:foo"
        xmlns:myext="urn:example:extensions">
  <namespace uri="urn:example:foo"/>
  <prefix value="foo"/>
  <import module="example-extensions">
    <prefix value="myext"/>
  </import>
  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

14. YANG ABNF 文法

YANG では、ほとんどのステートメントが順不同である。　ABNF 文法 [RFC 5234] [RFC 7405] 正規順序を定義する。　モジュールの可読性を向上させるために、この順序で節を入力することが推奨される(RECOMMENDED)。

ABNF 文法では、順序付けされていないステートメントはコメントでマークされる。

この文法では、スキャナーが YANG コメントを 1 つのスペース文字で置き換えることを前提としている。

```
 <CODE BEGINS> file "yang.abnf"
module-stmt         = optsep module-keyword sep identifier-arg-str
                      optsep
                      "{" stmtsep
                          module-header-stmts
                          linkage-stmts
                          meta-stmts
                          revision-stmts
                          body-stmts
                      "}" optsep

submodule-stmt      = optsep submodule-keyword sep identifier-arg-str
                      optsep
                      "{" stmtsep
                          submodule-header-stmts
                          linkage-stmts
                          meta-stmts
                          revision-stmts
                          body-stmts
                      "}" optsep

module-header-stmts = ;; these stmts can appear in any order
                      yang-version-stmt
                      namespace-stmt
                      prefix-stmt

submodule-header-stmts =
                      ;; these stmts can appear in any order
                      yang-version-stmt
                      belongs-to-stmt

meta-stmts          = ;; these stmts can appear in any order
                       [organization-stmt]
                      [contact-stmt]
                      [description-stmt]
```

```
                        [reference-stmt]

linkage-stmts       = ;; these stmts can appear in any order
                        *import-stmt
                        *include-stmt

revision-stmts      = *revision-stmt

body-stmts          = *(extension-stmt /
                          feature-stmt /
                          identity-stmt /
                          typedef-stmt /
                          grouping-stmt /
                          data-def-stmt /
                          augment-stmt /
                          rpc-stmt /
                          notification-stmt /
                          deviation-stmt)

data-def-stmt       = container-stmt /
                        leaf-stmt /
                        leaf-list-stmt /
                        list-stmt /
                        choice-stmt /
                        anydata-stmt /
                        anyxml-stmt /
                        uses-stmt

yang-version-stmt   = yang-version-keyword sep yang-version-arg-str
                        stmtend

yang-version-arg-str = < a string that matches the rule >
                         < yang-version-arg >

yang-version-arg    = "1.1"

import-stmt         = import-keyword sep identifier-arg-str optsep
                        "{" stmtsep
                            ;; these stmts can appear in any order
                            prefix-stmt
                            [revision-date-stmt]
                           [description-stmt]
                           [reference-stmt]
```

```
                    "}" stmtsep

include-stmt        = include-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [revision-date-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

namespace-stmt      = namespace-keyword sep uri-str stmtend
uri-str             = < a string that matches the rule >
                       < URI in RFC 3986 >

prefix-stmt         = prefix-keyword sep prefix-arg-str stmtend

belongs-to-stmt     = belongs-to-keyword sep identifier-arg-str
                       optsep
                       "{" stmtsep
                           prefix-stmt
                       "}" stmtsep

organization-stmt   = organization-keyword sep string stmtend

contact-stmt        = contact-keyword sep string stmtend

description-stmt    = description-keyword sep string stmtend

reference-stmt      = reference-keyword sep string stmtend

units-stmt          = units-keyword sep string stmtend

revision-stmt       = revision-keyword sep revision-date optsep
                       (";" /
                       "{" stmtsep
                            ;; these stmts can appear in any order
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

revision-date       = date-arg-str
```

```
revision-date-stmt  = revision-date-keyword sep revision-date stmtend

 extension-stmt      = extension-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [argument-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

argument-stmt       = argument-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           [yin-element-stmt]
                        "}") stmtsep

yin-element-stmt    = yin-element-keyword sep yin-element-arg-str
                       stmtend

yin-element-arg-str = < a string that matches the rule >
                       < yin-element-arg >

yin-element-arg     = true-keyword / false-keyword

identity-stmt       = identity-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *base-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

base-stmt           = base-keyword sep identifier-ref-arg-str
                       stmtend

feature-stmt        = feature-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
```

```
                          ;; these stmts can appear in any order
                         *if-feature-stmt
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                       "}") stmtsep

if-feature-stmt     = if-feature-keyword sep if-feature-expr-str
                       stmtend

if-feature-expr-str = < a string that matches the rule >
                       < if-feature-expr >

if-feature-expr     = if-feature-term
                        [sep or-keyword sep if-feature-expr]

if-feature-term     = if-feature-factor
                         [sep and-keyword sep if-feature-term]

if-feature-factor   = not-keyword sep if-feature-factor /
                       "(" optsep if-feature-expr optsep ")" /
                       identifier-ref-arg

typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           type-stmt
                           [units-stmt]
                          [default-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                        "}" stmtsep

type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                       (";" /
                       "{" stmtsep
                           [type-body-stmts]
                       "}") stmtsep

type-body-stmts     = numerical-restrictions /
                       decimal64-specification /
                       string-restrictions /
```

```
                        enum-specification /
                        leafref-specification /
                        identityref-specification /
                        instance-identifier-specification /
                        bits-specification /
                        union-specification /
                        binary-specification

 numerical-restrictions = [range-stmt]

range-stmt          = range-keyword sep range-arg-str optsep
                        (";" /
                        "{" stmtsep
                            ;; these stmts can appear in any order
                            [error-message-stmt]
                            [error-app-tag-stmt]
                            [description-stmt]
                            [reference-stmt]
                          "}") stmtsep

decimal64-specification = ;; these stmts can appear in any order
                            fraction-digits-stmt
                            [range-stmt]

fraction-digits-stmt = fraction-digits-keyword sep
                         fraction-digits-arg-str stmtend

fraction-digits-arg-str = < a string that matches the rule >
                           < fraction-digits-arg >

fraction-digits-arg = ("1" ["0" / "1" / "2" / "3" / "4" /
                              "5" / "6" / "7" / "8"])
                        / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

string-restrictions = ;; these stmts can appear in any order
                         [length-stmt]
                        *pattern-stmt

length-stmt         = length-keyword sep length-arg-str optsep
                        (";" /
                        "{" stmtsep
                            ;; these stmts can appear in any order
                            [error-message-stmt]
```

```
                            [error-app-tag-stmt]
                            [description-stmt]
                            [reference-stmt]
                          "}") stmtsep

pattern-stmt        = pattern-keyword sep string optsep
                        (";" /
                        "{" stmtsep
                            ;; these stmts can appear in any order
                            [modifier-stmt]
                            [error-message-stmt]
                            [error-app-tag-stmt]
                            [description-stmt]
                            [reference-stmt]
                          "}") stmtsep

modifier-stmt       = modifier-keyword sep modifier-arg-str stmtend

modifier-arg-str    = < a string that matches the rule >
                        < modifier-arg >

modifier-arg        = invert-match-keyword

default-stmt        = default-keyword sep string stmtend

enum-specification  = 1*enum-stmt

enum-stmt           = enum-keyword sep string optsep
                        (";" /
                        "{" stmtsep
                            ;; these stmts can appear in any order
                            *if-feature-stmt
                            [value-stmt]
                            [status-stmt]
                            [description-stmt]
                            [reference-stmt]
                          "}") stmtsep

leafref-specification =
                        ;; these stmts can appear in any order
                        path-stmt
                        [require-instance-stmt]
```

```
path-stmt           = path-keyword sep path-arg-str stmtend

require-instance-stmt = require-instance-keyword sep
                          require-instance-arg-str stmtend

require-instance-arg-str = < a string that matches the rule >
                            < require-instance-arg >

require-instance-arg = true-keyword / false-keyword

 instance-identifier-specification =
                       [require-instance-stmt]

identityref-specification =
                       1*base-stmt

union-specification = 1*type-stmt

binary-specification = [length-stmt]

bits-specification  = 1*bit-stmt

bit-stmt            = bit-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           [position-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

position-stmt       = position-keyword sep
                       position-value-arg-str stmtend

position-value-arg-str = < a string that matches the rule >
                          < position-value-arg >

position-value-arg  = non-negative-integer-value

status-stmt         = status-keyword sep status-arg-str stmtend
```

```
status-arg-str      = < a string that matches the rule >
                      < status-arg >

status-arg          = current-keyword /
                      obsolete-keyword /
                       deprecated-keyword

config-stmt         = config-keyword sep
                       config-arg-str stmtend

config-arg-str      = < a string that matches the rule >
                       < config-arg >

config-arg          = true-keyword / false-keyword

mandatory-stmt      = mandatory-keyword sep
                       mandatory-arg-str stmtend

mandatory-arg-str   = < a string that matches the rule >
                       < mandatory-arg >

mandatory-arg       = true-keyword / false-keyword

presence-stmt       = presence-keyword sep string stmtend
ordered-by-stmt     = ordered-by-keyword sep
                       ordered-by-arg-str stmtend

ordered-by-arg-str  = < a string that matches the rule >
                       < ordered-by-arg >

ordered-by-arg      = user-keyword / system-keyword

must-stmt           = must-keyword sep string optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                         "}") stmtsep

error-message-stmt  = error-message-keyword sep string stmtend
```

```
error-app-tag-stmt  = error-app-tag-keyword sep string stmtend

min-elements-stmt   = min-elements-keyword sep
                       min-value-arg-str stmtend

min-value-arg-str   = < a string that matches the rule >
                       < min-value-arg >

min-value-arg       = non-negative-integer-value

max-elements-stmt   = max-elements-keyword sep
                       max-value-arg-str stmtend

max-value-arg-str   = < a string that matches the rule >
                       < max-value-arg >

max-value-arg       = unbounded-keyword /
                      positive-integer-value

 value-stmt          = value-keyword sep integer-value-str stmtend

integer-value-str   = < a string that matches the rule >
                       < integer-value >

grouping-stmt       = grouping-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                           *action-stmt
                           *notification-stmt
                        "}") stmtsep

container-stmt      = container-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
```

```
                        *if-feature-stmt
                        *must-stmt
                        [presence-stmt]
                        [config-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        *data-def-stmt
                        *action-stmt
                        *notification-stmt
                     "}") stmtsep

leaf-stmt          = leaf-keyword sep identifier-arg-str optsep
                     "{" stmtsep
                        ;; these stmts can appear in any order
                        [when-stmt]
                         *if-feature-stmt
                        type-stmt
                         [units-stmt]
                        *must-stmt
                        [default-stmt]
                        [config-stmt]
                        [mandatory-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                      "}" stmtsep

leaf-list-stmt     = leaf-list-keyword sep identifier-arg-str optsep
                     "{" stmtsep
                         ;; these stmts can appear in any order
                        [when-stmt]
                         *if-feature-stmt
                        type-stmt stmtsep
                         [units-stmt]
                        *must-stmt
                        *default-stmt
                        [config-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                        [ordered-by-stmt]
                        [status-stmt]
```

```
                        [description-stmt]
                        [reference-stmt]
                      "}" stmtsep

list-stmt           = list-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                        ;; these stmts can appear in any order
                        [when-stmt]
                        *if-feature-stmt
                        *must-stmt
                        [key-stmt]
                        *unique-stmt
                        [config-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                        [ordered-by-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        1*data-def-stmt
                        *action-stmt
                        *notification-stmt
                       "}" stmtsep

key-stmt            = key-keyword sep key-arg-str stmtend

key-arg-str         = < a string that matches the rule >
                      < key-arg >

key-arg             = node-identifier *(sep node-identifier)

unique-stmt         = unique-keyword sep unique-arg-str stmtend

unique-arg-str      = < a string that matches the rule >
                      < unique-arg >

unique-arg          = descendant-schema-nodeid
                      *(sep descendant-schema-nodeid)

 choice-stmt         = choice-keyword sep identifier-arg-str optsep
                      (";" /
                      "{" stmtsep
```

```
                        ;; these stmts can appear in any order
                        [when-stmt]
                        *if-feature-stmt
                        [default-stmt]
                        [config-stmt]
                        [mandatory-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(short-case-stmt / case-stmt)
                     "}") stmtsep

short-case-stmt     = choice-stmt /
                      container-stmt /
                       leaf-stmt /
                       leaf-list-stmt /
                       list-stmt /
                       anydata-stmt /
                       anyxml-stmt

case-stmt           = case-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *data-def-stmt
                       "}") stmtsep

anydata-stmt        = anydata-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
```

```
                        [reference-stmt]
                      "}") stmtsep

anyxml-stmt         = anyxml-keyword sep identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                         "}") stmtsep

uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *refine-stmt
                           *uses-augment-stmt
                       "}") stmtsep

refine-stmt         = refine-keyword sep refine-arg-str optsep
                        "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *must-stmt
                           [presence-stmt]
                           *default-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [min-elements-stmt]
                           [max-elements-stmt]
                           [description-stmt]
                           [reference-stmt]
```

```
                        "}" stmtsep

refine-arg-str      = < a string that matches the rule >
                      < refine-arg >

refine-arg          = descendant-schema-nodeid

uses-augment-stmt   = augment-keyword sep uses-augment-arg-str optsep
                      "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                          1*(data-def-stmt / case-stmt /
                             action-stmt / notification-stmt)
                       "}" stmtsep

uses-augment-arg-str = < a string that matches the rule >
                       < uses-augment-arg >

uses-augment-arg    = descendant-schema-nodeid

augment-stmt        = augment-keyword sep augment-arg-str optsep
                      "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                          1*(data-def-stmt / case-stmt /
                             action-stmt / notification-stmt)
                       "}" stmtsep

augment-arg-str     = < a string that matches the rule >
                      < augment-arg >

augment-arg         = absolute-schema-nodeid

when-stmt           = when-keyword sep string optsep
                      (";" /
```

```
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        [description-stmt]
                        [reference-stmt]
                      "}") stmtsep

rpc-stmt            = rpc-keyword sep identifier-arg-str optsep
                    (";" /
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        *if-feature-stmt
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        [input-stmt]
                        [output-stmt]
                    "}") stmtsep

action-stmt         = action-keyword sep identifier-arg-str optsep
                    (";" /
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        *if-feature-stmt
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        [input-stmt]
                        [output-stmt]
                     "}") stmtsep

input-stmt          = input-keyword optsep
                    "{" stmtsep
                       ;; these stmts can appear in any order
                       *must-stmt
                       *(typedef-stmt / grouping-stmt)
                       1*data-def-stmt
                    "}" stmtsep

output-stmt         = output-keyword optsep
                    "{" stmtsep
                       ;; these stmts can appear in any order
```

```
                        *must-stmt
                        *(typedef-stmt / grouping-stmt)
                        1*data-def-stmt
                      "}" stmtsep

notification-stmt   = notification-keyword sep
                      identifier-arg-str optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *must-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                       "}") stmtsep

deviation-stmt      = deviation-keyword sep
                       deviation-arg-str optsep
                       "{" stmtsep
                          ;; these stmts can appear in any order
                          [description-stmt]
                          [reference-stmt]
                          (deviate-not-supported-stmt /
                             1*(deviate-add-stmt /
                                deviate-replace-stmt /
                                deviate-delete-stmt))
                       "}" stmtsep

deviation-arg-str   = < a string that matches the rule >
                       < deviation-arg >

deviation-arg       = absolute-schema-nodeid

deviate-not-supported-stmt =
                       deviate-keyword sep
                       not-supported-keyword-str stmtend

deviate-add-stmt    = deviate-keyword sep add-keyword-str optsep
                       (";" /
                       "{" stmtsep
```

```
                        ;; these stmts can appear in any order
                        [units-stmt]
                        *must-stmt
                        *unique-stmt
                        *default-stmt
                        [config-stmt]
                        [mandatory-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                      "}") stmtsep

deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep
                      (";" /
                      "{" stmtsep
                        ;; these stmts can appear in any order
                        [units-stmt]
                        *must-stmt
                        *unique-stmt
                        *default-stmt
                      "}") stmtsep

deviate-replace-stmt = deviate-keyword sep replace-keyword-str optsep
                      (";" /
                      "{" stmtsep
                        ;; these stmts can appear in any order
                        [type-stmt]
                        [units-stmt]
                        [default-stmt]
                        [config-stmt]
                        [mandatory-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                      "}") stmtsep

not-supported-keyword-str = < a string that matches the rule >
                            < not-supported-keyword >

add-keyword-str     = < a string that matches the rule >
                      < add-keyword >

delete-keyword-str  = < a string that matches the rule >
                      < delete-keyword >
```

```
replace-keyword-str = < a string that matches the rule >
                      < replace-keyword >

;; represents the usage of an extension

 unknown-statement   = prefix ":" identifier [sep string] optsep
                       (";" /
                       "{" optsep
                           *((yang-stmt / unknown-statement) optsep)
                         "}") stmtsep

yang-stmt           = action-stmt /
                      anydata-stmt /
                      anyxml-stmt /
                      argument-stmt /
                      augment-stmt /
                      base-stmt /
                      belongs-to-stmt /
                      bit-stmt /
                      case-stmt /
                      choice-stmt /
                      config-stmt /
                      contact-stmt /
                      container-stmt /
                      default-stmt /
                      description-stmt /
                      deviate-add-stmt /
                      deviate-delete-stmt /
                      deviate-not-supported-stmt /
                      deviate-replace-stmt /
                      deviation-stmt /
                      enum-stmt /
                      error-app-tag-stmt /
                      error-message-stmt /
                      extension-stmt /
                     feature-stmt /
                      fraction-digits-stmt /
                      grouping-stmt /
                      identity-stmt /
                      if-feature-stmt /
                      import-stmt /
                      include-stmt /
                      input-stmt /
```

```
                        key-stmt /
                        leaf-list-stmt /
                        leaf-stmt /
                        length-stmt /
                        list-stmt /
                        mandatory-stmt /
                        max-elements-stmt /
                        min-elements-stmt /
                        modifier-stmt /
                        module-stmt /
                        must-stmt /
                        namespace-stmt /
                        notification-stmt /
                        ordered-by-stmt /
                        organization-stmt /
                        output-stmt /
                        path-stmt /
                        pattern-stmt /
                        position-stmt /
                        prefix-stmt /
                        presence-stmt /
                        range-stmt /
                        reference-stmt /
                        refine-stmt /
                        require-instance-stmt /
                        revision-date-stmt /
                        revision-stmt /
                        rpc-stmt /
                        status-stmt /
                        submodule-stmt /
                        typedef-stmt /
                        type-stmt /
                       unique-stmt /
                        units-stmt /
                        uses-augment-stmt /
                        uses-stmt /
                       value-stmt /
                        when-stmt /
                        yang-version-stmt /
                        yin-element-stmt

;; Ranges
```

```
range-arg-str       = < a string that matches the rule >
                      < range-arg >

range-arg           = range-part *(optsep "|" optsep range-part)

range-part          = range-boundary
                      [optsep ".." optsep range-boundary]

range-boundary      = min-keyword / max-keyword /
                      integer-value / decimal-value

 ;; Lengths

length-arg-str      = < a string that matches the rule >
                      < length-arg >

length-arg          = length-part *(optsep "|" optsep length-part)

length-part         = length-boundary
                      [optsep ".." optsep length-boundary]

length-boundary     = min-keyword / max-keyword /
                      non-negative-integer-value

 ;; Date

date-arg-str        = < a string that matches the rule >
                      < date-arg >

date-arg            = 4DIGIT "-" 2DIGIT "-" 2DIGIT

 ;; Schema Node Identifiers

schema-nodeid       = absolute-schema-nodeid /
                      descendant-schema-nodeid

absolute-schema-nodeid = 1*("/" node-identifier)

 descendant-schema-nodeid =
                      node-identifier
                      [absolute-schema-nodeid]

node-identifier     = [prefix ":"] identifier
```

```
 ;; Instance Identifiers

instance-identifier = 1*("/" (node-identifier
                              [1*key-predicate /
                               leaf-list-predicate /
                               pos]))

key-predicate       = "[" *WSP key-predicate-expr *WSP "]"

key-predicate-expr  = node-identifier *WSP "=" *WSP quoted-string

leaf-list-predicate = "[" *WSP leaf-list-predicate-expr *WSP "]"

leaf-list-predicate-expr = "." *WSP "=" *WSP quoted-string

pos                 = "[" *WSP positive-integer-value *WSP "]"

quoted-string       = (DQUOTE string DQUOTE) / (SQUOTE string SQUOTE)

 ;; leafref path

path-arg-str        = < a string that matches the rule >
                       < path-arg >

path-arg            = absolute-path / relative-path

absolute-path       = 1*("/" (node-identifier *path-predicate))

relative-path       = 1*("../") descendant-path

descendant-path     = node-identifier
                       [*path-predicate absolute-path]

path-predicate      = "[" *WSP path-equality-expr *WSP "]"

path-equality-expr  = node-identifier *WSP "=" *WSP path-key-expr

path-key-expr       = current-function-invocation *WSP "/" *WSP
                       rel-path-keyexpr

rel-path-keyexpr    = 1*(".." *WSP "/" *WSP)
                       *(node-identifier *WSP "/" *WSP)
```

```
                        node-identifier
 ;;; Keywords, using the syntax for case-sensitive strings (RFC 7405)

;; statement keywords

 action-keyword           = %s"action"
 anydata-keyword          = %s"anydata"
 anyxml-keyword           = %s"anyxml"
 argument-keyword         = %s"argument"
 augment-keyword          = %s"augment"
 base-keyword             = %s"base"
 belongs-to-keyword       = %s"belongs-to"
 bit-keyword              = %s"bit"
 case-keyword             = %s"case"
 choice-keyword           = %s"choice"
 config-keyword           = %s"config"
 contact-keyword          = %s"contact"
 container-keyword        = %s"container"
 default-keyword          = %s"default"
 description-keyword      = %s"description"
 deviate-keyword          = %s"deviate"
 deviation-keyword        = %s"deviation"
 enum-keyword             = %s"enum"
 error-app-tag-keyword    = %s"error-app-tag"
 error-message-keyword    = %s"error-message"
 extension-keyword        = %s"extension"
 feature-keyword          = %s"feature"
 fraction-digits-keyword  = %s"fraction-digits"
 grouping-keyword         = %s"grouping"
 identity-keyword         = %s"identity"
 if-feature-keyword       = %s"if-feature"
 import-keyword           = %s"import"
 include-keyword          = %s"include"
 input-keyword            = %s"input"
 key-keyword              = %s"key"
 leaf-keyword             = %s"leaf"
 leaf-list-keyword        = %s"leaf-list"
 length-keyword           = %s"length"
 list-keyword             = %s"list"
 mandatory-keyword        = %s"mandatory"
 max-elements-keyword     = %s"max-elements"
 min-elements-keyword     = %s"min-elements"
 modifier-keyword         = %s"modifier"
```

```
 module-keyword           = %s"module"
 must-keyword             = %s"must"
 namespace-keyword        = %s"namespace"
 notification-keyword     = %s"notification"
 ordered-by-keyword       = %s"ordered-by"
 organization-keyword     = %s"organization"
 output-keyword           = %s"output"
 path-keyword             = %s"path"
 pattern-keyword          = %s"pattern"
 position-keyword         = %s"position"
 prefix-keyword           = %s"prefix"
 presence-keyword         = %s"presence"
 range-keyword            = %s"range"
 reference-keyword        = %s"reference"
 refine-keyword           = %s"refine"
 require-instance-keyword = %s"require-instance"
 revision-keyword         = %s"revision"
 revision-date-keyword    = %s"revision-date"
 rpc-keyword              = %s"rpc"
 status-keyword           = %s"status"
 submodule-keyword        = %s"submodule"
 type-keyword             = %s"type"
 typedef-keyword          = %s"typedef"
 unique-keyword           = %s"unique"
 units-keyword            = %s"units"
 uses-keyword             = %s"uses"
 value-keyword            = %s"value"
 when-keyword             = %s"when"
 yang-version-keyword     = %s"yang-version"
 yin-element-keyword      = %s"yin-element"

 ;; other keywords

add-keyword              = %s"add"
 current-keyword          = %s"current"
 delete-keyword           = %s"delete"
 deprecated-keyword       = %s"deprecated"
 false-keyword            = %s"false"
 invert-match-keyword     = %s"invert-match"
 max-keyword              = %s"max"
 min-keyword              = %s"min"
not-supported-keyword    = %s"not-supported"
 obsolete-keyword         = %s"obsolete"
```

```
    replace-keyword          = %s"replace"
    system-keyword           = %s"system"
    true-keyword             = %s"true"
    unbounded-keyword        = %s"unbounded"
    user-keyword             = %s"user"
    and-keyword              = %s"and"
    or-keyword               = %s"or"
    not-keyword              = %s"not"

   current-function-invocation = current-keyword *WSP "(" *WSP ")"    ;;; Basic
Rules

   prefix-arg-str      = < a string that matches the rule >
                         < prefix-arg >
   prefix-arg          = prefix
   prefix              = identifier
   identifier-arg-str  = < a string that matches the rule >
                         < identifier-arg >
   identifier-arg      = identifier
   identifier          = (ALPHA / "_")
                         *(ALPHA / DIGIT / "_" / "-" / ".")
   identifier-ref-arg-str = < a string that matches the rule >
                            < identifier-ref-arg >
   identifier-ref-arg  = identifier-ref
   identifier-ref      = [prefix ":"] identifier
   string              = < an unquoted string, as returned by >
                         < the scanner, that matches the rule >
                         < yang-string >
   yang-string         = *yang-char

   ;; any Unicode or ISO/IEC 10646 character, including tab, carriage
   ;; return, and line feed but excluding the other C0 control
    ;; characters, the surrogate blocks, and the noncharacters
    yang-char = %x09 / %x0A / %x0D / %x20-D7FF /
                               ; exclude surrogate blocks %xD800-DFFF
              %xE000-FDCF /    ; exclude noncharacters %xFDD0-FDEF
              %xFDF0-FFFD /    ; exclude noncharacters %xFFFE-FFFF
              %x10000-1FFFD /  ; exclude noncharacters %x1FFFE-1FFFF
              %x20000-2FFFD /  ; exclude noncharacters %x2FFFE-2FFFF
              %x30000-3FFFD /  ; exclude noncharacters %x3FFFE-3FFFF
              %x40000-4FFFD /  ; exclude noncharacters %x4FFFE-4FFFF
              %x50000-5FFFD /  ; exclude noncharacters %x5FFFE-5FFFF
              %x60000-6FFFD /  ; exclude noncharacters %x6FFFE-6FFFF
```

```
            %x70000-7FFFD /  ; exclude noncharacters %x7FFFE-7FFFF
            %x80000-8FFFD /  ; exclude noncharacters %x8FFFE-8FFFF
            %x90000-9FFFD /  ; exclude noncharacters %x9FFFE-9FFFF
            %xA0000-AFFFD /  ; exclude noncharacters %xAFFFE-AFFFF
            %xB0000-BFFFD /  ; exclude noncharacters %xBFFFE-BFFFF
            %xC0000-CFFFD /  ; exclude noncharacters %xCFFFE-CFFFF
            %xD0000-DFFFD /  ; exclude noncharacters %xDFFFE-DFFFF
            %xE0000-EFFFD /  ; exclude noncharacters %xEFFFE-EFFFF
            %xF0000-FFFFD /  ; exclude noncharacters %xFFFFE-FFFFF
             %x100000-10FFFD  ; exclude noncharacters %x10FFFE-10FFFF

integer-value       = ("-" non-negative-integer-value) /
                        non-negative-integer-value

non-negative-integer-value = "0" / positive-integer-value
positive-integer-value = (non-zero-digit *DIGIT)
zero-integer-value  = 1*DIGIT
stmtend             = optsep (";" / "{" stmtsep "}") stmtsep
sep                 = 1*(WSP / line-break)
                       ; unconditional separator
optsep              = *(WSP / line-break)
stmtsep             = *(WSP / line-break / unknown-statement)
line-break          = CRLF / LF
non-zero-digit      = %x31-39
decimal-value       = integer-value ("." zero-integer-value)
SQUOTE              = %x27
                      ; single quote

;;; core rules from RFC 5234

ALPHA               = %x41-5A / %x61-7A
                      ; A-Z / a-z
CR                  = %x0D
                      ; carriage return
CRLF                = CR LF
                      ; Internet standard newline
DIGIT               = %x30-39
                      ; 0-9
DQUOTE              = %x22
                      ; double quote
HTAB                = %x09
                      ; horizontal tab
LF                  = %x0A
```

```
                    ; line feed
SP                  = %x20
                    ; space
WSP                 = SP / HTAB
                    ; whitespace
<CODE ENDS>
```

15. YANG 関連エラーに対する NETCONF エラー応答

データモデルの処理に関連するエラーケースに対して、多数の NETCONF エラー応答が定義される。該当する YANG ステートメントに"error-app-tag"サブステートメントがある場合は、以下で指定されているデフォルト値が上書きされる。

15.1. "unique"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、設定データで"unique"制約が無効になる場合、以下のエラーを返さなければならない(MUST)。

```
error-tag:      operation-failed
error-app-tag:  data-not-unique
error-info:     <non-unique>: Contains an instance identifier that
                points to a leaf that invalidates the "unique"
                constraint.  This element is present once for each
                non-unique leaf.
                The <non-unique> element is in the YANG
                namespace ("urn:ietf:params:xml:ns:yang:1").
```

15.2. "max-elements"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、リストまたはリーフリストのエントリ数が多すぎる設定データが得られた場合、以下のエラーを返さなければならない(MUST)。

```
error-tag:      operation-failed
error-app-tag:  too-many-elements
```

このエラーは、追加の子ノードが複数存在する場合でも、リストノードを識別するエラーパスとともに 1 回返される。

15.3. "min-elements"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、リストまたはリーフリストのエントリ数が少なすぎる設定データになった場合、以下のエラーを返さなければならない(MUST)。

```
error-tag:      operation-failed
error-app-tag:  too-few-elements
```

このエラーは、複数の子ノードが欠落している場合でも、リストノードを識別するエラーパスとともに 1 回返される。

15.4. "must"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、"must"ステートメントによって課せられた制限に違反する設定データになった場合、特定の"error-app-tag"サブステートメントが"must"ステートメントに存在しない限り、以下のエラーを返さなければならない(MUST)。

```
error-tag:      operation-failed
```

```
error-app-tag:  must-violation
```

15.5. "require-instance"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、"instance-identifier"または"leafref"型で require-instance が"true"でマークされたリーフが存在しないインスタンスを参照する設定データになった場合、以下のエラーを返さなければならない(MUST)。

```
error-tag:      data-missing
error-app-tag:  instance-required
error-path:     Path to the instance-identifier or leafref leaf.
```

15.6. 必須の"choice"ステートメントに違反するデータのエラーメッセージ

NETCONF 操作の結果、必須の choice にノードが存在しない設定データになった場合、以下のエラーを返さなければならない(MUST)。

```
error-tag:      data-missing
error-app-tag:  missing-choice
error-path:     Path to the element with the missing choice.
error-info:     <missing-choice>: Contains the name of the missing
                 mandatory choice.
                The <missing-choice> element is in the YANG
                namespace ("urn:ietf:params:xml:ns:yang:1").
```

15.7. "insert"操作のエラーメッセージ

"insert"および"key"または"value"属性がリストまたはリーフリストノードの<edit-config>で使用され、"key"または"value"が存在しないインスタンスを参照している場合は、以下のエラーを返さなければならない(MUST)。

```
error-tag:      bad-attribute
error-app-tag:  missing-instance
```

16. IANA の考慮事項

この文書は、"Network Configuration Protocol (NETCONF) Capability URN"レジストリから一つの機能識別子 URN を登録する。

Index	Capability Identifier
:yang-library	urn:ietf:params:netconf:capability:yang-library:1.0

17. セキュリティに関する考慮事項

このドキュメントでは、管理情報の説明を記述および読み取るための言語を定義する。　言語自体は、インターネットのセキュリティに影響を与えない。

基本的には NETCONF プロトコルと同じ考慮事項が適用される([RFC 6241] のセクション 9 参照)。

YANG でモデル化されたデータには、機密情報が含まれている可能性がある。　YANG で定義された RPC または通知は、機密情報を転送する場合がある。

セキュリティの問題は、YANG でモデル化されたデータの利用に関連しており、そのような問題は、データモデルを記述した文書と、データを操作するために使用されるインターフェイスに関する文書、例えば NETCONF 文書で扱われなければならない(MUST)。　YANG でモデル化されるデータは、次の要素に依存する。

- 機密情報の送信に使用される送信インフラストラクチャのセキュリティ。
- そのような機密情報を格納またはリリースするアプリケーションのセキュリティ。
- 機密データの使用を制限するための適切な認証およびアクセス制御メカニズム。

YANG パーサは、不正な文書に対して堅牢である必要がある。　未知または信頼できないソースから不正な形式のドキュメントを読み取ると、攻撃者が YANG パーサを実行しているユーザの権限を取得する可能性がある。　極端な状況では、マシン全体が危険にさらされる可能性がある。

18. 参考資料

18.1. 規格参照

[ISO.10646]

International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[XML-NAMES] Bray, T., Hollander, D., Layman, A., Tobin, R., and H.Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

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

[XSD-TYPES] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

18.2. 参考情報

[ISO.10646]

International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[XML-NAMES] Bray, T., Hollander, D., Layman, A., Tobin, R., and H.Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

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

[XSD-TYPES] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.