Internet Engineering Task Force (IETF) M. Bjorklund, Ed.
Request for Comments: 6020 Tail-f Systems
Category: Standards Track October 2010
ISSN: 2070-1721

YANG - моделирование данных для протокола NETCONF

YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)

Аннотация

YANG представляет собой язык моделирования данных, используемый в моделях конфигурации и состояний, с которыми работает протокол настройки конфигурации сети (Network Configuration Protocol или NETCONF), вызовы удалённых процедур NETCONF и уведомления NETCONF.

Статус документа

Этот документ является проектом стандарта (Internet Standards Track).

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительная информация о документах Internet Standard представлена в разделе 2 документа RFC 5741.

Информация о статусе этого документа, обнаруженных ошибках и способах обратной связи доступна по ссылке http://www.rfc-editor.org/info/rfc6020.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

YANG представляет собой язык моделирования данных, используемый в моделях конфигурации и состояний, с которыми работает протокол настройки конфигурации сети (NETCONF), вызовы удалённых процедур NETCONF и уведомления NETCONF. YANG служит для моделирования операций и содержимого уровней NETCONF (см. параграф 1.1 в [RFC4741]).

Этот документ описывает синтаксис и семантику языка YANG, представления модели данных, определённой в модуле YANG на языке XML[1], и использование операций NETCONF для манипулирования данными.

2. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119].

3. Терминология

anyxml
Узел данных, содержащий неизвестный блок (chunk) данных XML.

augment - дополнение
Добавляет новые узлы схемы к ранее определённому узлу.

base type - базовый тип
Тип, на основе которого создан производный тип. Может быть встроенным или производным типом.

built-in type - встроенный тип
Тип данных YANG, определённый в спецификации языка YANG (например, uint32 или string).

choice - выбор
Узел схема, где можно выбрать какой-либо один из имеющихся вариантов.

configuration data - данные конфигурации
Набор данных (с возможностью записи), который требуется для перевода системы из исходного состояния в текущее [RFC4741].

conformance - соответствие
Мера соответствия устройства модели данных.

container - контейнер
Внутренний узел данных, имеющийся в дереве данных не более чем в одном экземпляре. Контейнер не имеет значения и представляет собой набор дочерних узлов.

data definition statement - оператор определения данных
Оператор, определяющий новые узлы данных, - container, leaf, leaf-list, list, choice, case, augment, uses и anyxml.

data model - модель данных
Модель, описывающая представление данных и доступ к ним.

data node - узел данных
Узел в дереве схемы, экземпляр которого может быть создан в дереве данных, - container, leaf, leaf-list, list, anyxml.

[1]Extensible Markup Language - расширяемый язык разметки.

data tree - дерево данных
Экземпляр дерева данных конфигурации и состояния в устройстве.

derived type - производный тип
Тип, созданный на основе встроенного (например, uint32) или другого производного типа.

device deviation - отклонение устройства
Отказ устройства корректно реализовать модуль.

extension - расширение
Расширения добавляют в операторы семантику, отличную от YANG. Оператор extension определяет новые операторы для выражения такой семантики.

feature - возможность, функция
Механизм для маркировки части модели как необязательной. Определения могут помечаться именами возможностей и будут применимы только на устройствах, поддерживающих соответствующую возможность.

grouping - группировка
Набор многократно используемых (reusable) узлов схемы, который может использоваться локально в модуле или в других модулях, импортирующих его. Оператор grouping не является оператором определения данных, поэтому он не задаёт каких-либо узлов в дереве схемы.

identifier - идентификатор
Строка, используемая для указания различных элементов YANG по именам.

instance identifier - идентификатор экземпляра
Механизм для идентификации отдельного узла в дереве данных.

interior node - внутренний узел
Узел в иерархии, который не является листом (leaf).

leaf - лист
Узел данных, который существует в дереве данных в количестве не более одного экземпляра. Лист имеет значение, но не имеет дочерних узлов.

leaf-list - лист-список
Похож на лист, но определяет набор однозначно идентифицируемых узлов, а не один узел. Каждый из узлов списка имеет значение, но не имеет узлов-потомков.

list - список
Внутренний узел данных, который может существовать в дереве данных во множестве экземпляров. Список не имеет значения, но имеет набор дочерних узлов.

module - модуль
Модуль YANG определяет иерархию узлов, которые могут использоваться для операций на основе NETCONF. Со своими и импортированными или включёнными из любого источника определениями модуль является самодостаточным и «компилируемым».

RPC[1]
Вызов удалённой процедуры, используемый в протоколе NETCONF.

RPC operation - операция RPC
Вызов конкретной удалённой процедуры, используемый в протоколе NETCONF. Называется также протокольной операцией.

schema node - узел схемы
Узел в дереве схемы - container, leaf, leaf-list, list, choice, case, rpc, input, output, notification или anyxml.

schema node identifier идентификатор узла схемы
Механизм для указания конкретного узла в дереве схемы.

schema tree - дерево схемы
Определение иерархии, заданное внутри модуля.

state data - данные состояния
Дополнительные данные в системе, которые не относятся к конфигурации, например, доступная лишь для чтения статистика и информация о состоянии [RFC4741].

submodule - субмодуль
Частичное определение модуля, вносящее производные типы, группировки, узлы данных, RPC, действия и уведомления для модуля. Модуль YANG может содержать множество субмодулей.

top-level data node - узел данных верхнего уровня
Узел данных, не имеющий других узлов данных между собой и оператором module или submodule.

uses - использует
Оператор служит для создания экземпляра множества узлов схемы, определённого в операторе grouping. Создаваемые узлы могут быть уточнены и дополнены в соответствии с конкретными потребностями.

3.1. Обязательные узлы

Обязательными узлами являются перечисленные ниже:

- узлы leaf, choice или anyxml с оператором mandatory, имеющим значение true;
- узлы list или leaf-list, в которых оператор min-elements имеет значение больше 0;
- узел container без оператора presence, который имеет хотя бы один обязательный узел в качестве потомка.

4. Обзор YANG

4.1. Функциональный обзор

Язык YANG служит для моделирования данных в протоколе NETCONF. Модуль YANG определяет иерархии данных, которые могут применяться в операциях на основе NETCONF, включая параметры конфигурации, данные состояния, RPC и уведомления. Это позволило полностью описать данные, передаваемые между клиентами и серверами NETCONF.

[1]Remote Procedure Call.

YANG моделирует иерархическую организацию данных в виде дерева, где каждый узел имеет имя и значение или набор дочерних узлов. YANG обеспечивает чёткое и краткое описание узлов, а также взаимодействия между ними.

YANG структурирует модели данных в модули и субмодули. Модуль может импортировать определения из внешних модулей и включать определения из субмодулей. Иерархия может дополняться путём позволения модулю добавлять узлы данных в иерархию, определённую в другом модуле. Дополнение может быть условным, когда новые узлы добавляются только при выполнении заданных условий.

Модели данных YANG могут описывать ограничения, применяемые к данным, ограничивая присутствие или значения узлов в зависимости от наличия или значения других узлов в иерархии. Эти ограничения применяются клиентом или сервером и корректное содержимое **должно** соблюдать ограничения.

YANG определяет набор встроенных типов и включает механизм для определения новых типов. Производные типы могут ограничивать набор значений по сравнению с базовым типом, путём применения механизмов типа ограничений на числа элементов или шаблонов (pattern), которые могут применяться клиентами или серверами. Они могут также задавать соглашения по использованию производного типа как, например, основанный на string тип для указания имён хостов.

YANG разрешает использовать группировку узлов для многократного применения (reusable). Создание экземпляра такой группировки позволяет уточнять или дополнять узлы, обеспечивая возможность адаптации узлов к конкретным потребностям. Производные типы и группировки могут определяться в одном модуле и применяться в нем же или в другом модуле или субмодуле, который импортирует или включает данный модуль

Конструкции иерархии данных YANG включают определения списков, которые идентифицируются ключами, отличающимися один от другого. Такие списки могут определяться как сортируемые пользователем или автоматически сортироваться системой. В сортируемых пользователем списках определяются операции для изменения порядка элементов в списке.

Модули YANG могут транслироваться в эквивалентный синтаксис XML, называемый YIN[1] (раздел 11), что позволяет приложениям использовать анализаторы XML и сценарии XSLT[2] для работы с моделями. Преобразование из YANG в YIN происходит без семантических потерь, поэтому YIN можно обратно преобразовать в YANG.

YANG обеспечивает баланс моделирования данных на верхнем уровне и кодирования битов на нижнем уровне. Читатель модуля YANG может видеть верхний уровень представления модели данных, понимая, как эти данные будут кодироваться в операциях NETCONF.

Язык YANG является расширяемым и позволяет определять расширения органам стандартизации, производителям и отдельным людям. Синтаксис операторов позволяет расширениям естественным образом существовать вместе со стандартными операторами, при этом расширения в модуле YANG достаточно выделяются, чтобы быть заметными.

YANG не пытается решить все возможные задачи, ограничивая область действия выражением моделей данных NETCONF, а не произвольных документов XML или моделей данных. Модели данных, описываемые языком YANG разработаны для упрощения работы с операциями NETCONF.

Насколько это возможно, YANG поддерживает совместимость со структурами SMIv2[3] [RFC2578] [RFC2579] протокола SNMP[4]. Основанные на SMIv2 модули MIB могут автоматически преобразовываться в модули YANG с доступом только для чтения (read-only). Однако YANG не обеспечивает трансляции в SMIv2.

Подобно NETCONF, YANG обеспечивает интеграцию с естественной инфраструктурой управления устройством. Это позволяет реализациям использовать имеющиеся механизмы контроля доступа для защиты или показа элементов модели данных.

4.2. Обзор языка

В этом разделе вводятся некоторые важные конструкции YANG, которые будут нужны для понимания специфики языка в последующих параграфах. Такой прогрессивный подход учитывает взаимосвязанную природу концепций и операторов YANG. Подробное описание операторов YANG и их синтаксиса начинается в разделе 7.

4.2.1. Модули и субмодули

Модули содержат три типа операторов: операторы заголовка, операторы версии (revision) и операторы определений. Операторы заголовка описывают модуль и дают информацию о самом модуле, операторы revision указывают сведения об истории модуля, а операторы определений являются телом модуля, где определяется модель данных.

Сервер NETCONF может поддерживать множество модулей, давая разные представления для одних и тех же данных или набор представлений отдельных подмножеств данных сервера. Сервер может также поддерживать единственный модуль, определяющий все доступные данные.

Модуль может разделить фрагменты своих данных на субмодули в соответствии с потребностями владельца модуля. Извне это будет по-прежнему выглядеть единым модулем, независимо от наличия и размера субмодулей.

Оператор include позволяет модулю или субмодулю ссылаться на свои субмодули, а оператор import позволяет ссылаться на данные в других модулях.

4.2.2. Основы моделирования данных

YANG определяет четыре основных типа узлов данных для моделирования данных. В каждом из последующих параграфов примеры показывают синтаксис YANG и соответствующее представление XML.

[1]YANG Independent Notation - независимая от YANG нотация.
[2]Extensible Stylesheet Language Transformation - преобразование расширяемого языка стилей.
[3]Structure of Management Information - структура данных управления.
[4]Simple Network Management Protocol - простой протокол управления сетями.

4.2.2.1. Лист

Узел leaf содержит простые данные типа integer или string, имеет единственное значение конкретного типа и не имеет дочерних узлов.

Пример YANG

```
leaf host-name {
    type string;
    description "Имя хоста для данной системы";
}
```

Пример NETCONF XML

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

Оператор leaf описан в параграфе 7.6.

4.2.2.2. Список листьев

Список листьев (leaf-list) определяет последовательность значений определённого типа с единственным значением конкретного типа для каждого листа.

Пример YANG

```
leaf-list domain-search {
    type string;
    description "Список доменных имён для поиска";
}
```

Пример NETCONF XML

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

Оператор leaf-list описан в параграфе 7.7.

4.2.2.3. Контейнер

Контейнеры служат для группировки связанных узлов в субдерево (ветвь). Контейнер имеет лишь дочерние узлы и не имеет значения. Контейнер может содержать любое число узлов произвольного типа (листья, списки, контейнеры, листья-списки).

Пример YANG

```
container system {
    container login {
        leaf message {
            type string;
            description
                "Сообщение при старте сеанса входа в систему";
        }
    }
}
```

Пример NETCONF XML

```
<system>
  <login>
    <message>Доброе утро</message>
  </login>
</system>
```

Оператор container описан в параграфе 7.5.

4.2.2.4. Список

Список определяет последовательность элементов. Каждый элемент подобен структуре или экземпляру записи и однозначно идентифицируется значениями ключевых листьев. Список может определять множество ключевых листьев и включать произвольное число дочерних узлов любого типа (листья, списки, контейнеры и т. п.).

Пример YANG

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

Пример NETCONF XML

```
<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>
```

Оператор list описан в параграфе 7.8.

4.2.2.5. Пример модуля

Ниже показан пример комбинации операторов для определения модуля.

```
// Содержимое примера "acme-system.yang"
module acme-system {
    namespace "http://acme.example.com/system";
    prefix "acme";

    organization "ACME Inc.";
    contact "joe@acme.example.com";
    description
        "Модуль для элементов, реализующих систему ACME.";

    revision 2007-06-09 {
        description "Первый выпуск.";
    }

    container system {
        leaf host-name {
            type string;
            description "Имя хоста для этой системы";
        }

        leaf-list domain-search {
            type string;
            description "Список доменных имён для поиска";
        }

        container login {
            leaf message {
                type string;
                description
                    "Сообщение, выдаваемое при старте сеанса входа";
            }

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

4.2.3. Данные состояния

YANG может моделировать данные состояния, а также конфигурационные данные в зависимости от оператора config. Когда узел имеет тег config false, это означает, что его субиерархия помечена, как данные состояния, возвращаемые операцией NETCONF <get>, но не <get-config>. Родительские контейнеры, списки и ключевые узлы также указываются, как содержащие контекст для данных состояния.

В приведённом ниже примере для каждого интерфейса определены два узла, определяющие заданную в конфигурации и наблюдаемую скорость. Наблюдаемая скорость не является конфигурационной, поскольку она может быть возвращена операцией NETCONF <get>, но не <get-config>. Наблюдаемая скорость не относится к данным конфигурации и её нельзя установить с помощью <edit-config>.

```
list interface {
    key "name";

    leaf name {
        type string;
    }
    leaf speed {
        type enumeration {
            enum 10m;
            enum 100m;
            enum auto;
        }
    }
    leaf observed-speed {
```

```
            type uint32;
            config false;
        }
    }
```

4.2.4. Встроенные типы

Язык YANG имеет набор встроенных типов, подобно многим языкам программирования, но с некоторыми отличиями, которые связаны со специальными требованиями сетевого управления. В таблице перечислены встроенные типы, которые подробно описаны в разделе 9.

Имя	*Описание*
binary	произвольные двоичные данные
bits	набор битов или флагов
boolean	true или false
decimal64	64-битовое десятичное число со знаком
empty	лист, не имеющий какого-либо значения
enumeration	одно из перечисляемого набора строковых значений
identityref	ссылка на абстрактное отождествление (identity)
instance-identifier	ссылка на узел дерева данных
int8	8-битовое целое число со знаком
int16	16-битовое целое число со знаком
int32	32-битовое целое число со знаком
int64	64-битовое целое число со знаком
leafref	ссылка на экземпляр листа
string	строка символов
uint8	8-битовое целое число без знака
uint16	16-битовое целое число без знака
uint32	32-битовое целое число без знака
uint64	64-битовое целое число без знака
union	выбор одного из входящих в объединение типов

Оператор type описан в параграфе 7.4.

4.2.5. Производные типы (typedef)

YANG позволяет определять производные (derived) типы на основе базовых с помощью оператора typedef. Базовым типом может быть встроенный или ранее определённый производный тип, что позволяет создавать иерархии производных типов.

Производные типы могут указываться в качестве аргумента операторов type.

Пример YANG

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

    leaf completed {
        type percent;
    }
```

Пример NETCONF XML

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

Оператор typedef описан в параграфе 7.3.

4.2.6. Многократно используемые группы узлов (grouping)

Группы узлов могут собираться в многократно применяемые наборы и помощью оператора grouping. Группировка определяет множество узлов, которое создаётся с помощью оператора uses.

```
    grouping target {
        leaf address {
            type inet:ip-address;
            description "Целевой адрес IP";
        }
        leaf port {
            type inet:port-number;
            description "Номер целевого порта";
        }
    }

    container peer {
        container destination {
            uses target;
        }
    }
```

Пример NETCONF XML

```
    <peer>
      <destination>
        <address>192.0.2.1</address>
        <port>830</port>
      </destination>
```

```
</peer>
```

Группировка может быть уточнена, что позволяет переопределять в ней некоторые операторы. В приведённом ниже примере уточняется описание (description).

```
container connection {
    container source {
        uses target {
            refine "address" {
                description "IP-адрес отправителя";
            }
            refine "port" {
                description "Номер порта отправителя";
            }
        }
    }
    container destination {
        uses target {
            refine "address" {
                description "IP-адрес получателя";
            }
            refine "port" {
                description "Номер порта получателя";
            }
        }
    }
}
```

Оператор grouping описан в параграфе 7.11.

4.2.7. Выбор

YANG позволяет модели данных разделять несовместимые узлы в разные варианты с помощью операторов choice и case. Оператор choice содержит набор операторов case, определяющих набор узлов схемы, которые не могут появляться вместе. Каждый case может содержать множество узлов, но каждый узел может появляться только в одном case в рамках оператора choice.

При создании элемента из одного case все элементы прочих case неявно удаляются. Устройство обрабатывает выполнение ограничений, предотвращающих несовместимости в текущей конфигурации.

Узлы choice и case появляются только в дереве схемы и не могут присутствовать в дереве данных или сообщениях NETCONF. Дополнительные уровни иерархии за пределами концептуальной схемы не требуются.

Пример YANG

```
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;
              }
          }
      }
  }
}
```

Пример NETCONF XML

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

Оператор choice описан в параграфе 7.9.

4.2.8. Расширение моделей данных (augment)

YANG позволяет вставлять дополнительные узлы в модель данных как текущего модуля (и его субмодулей), так и внешних модулей. Это полезно, например, для производителей, добавляющих свои параметры в стандартные модели данных с обеспечением совместимости.

Оператор augment определяет место в иерархии модели данных, куда помещаются новые узлы, а оператор when задаёт условия, когда новые узлы становятся пригодными (вступают в силу).

Пример YANG

```
augment /system/login/user {
    when "class != 'wheel'";
    leaf uid {
        type uint16 {
```

```
                range "1000 .. 30000";
            }
        }
    }
```

В этом примере определён узел uid, который начинает действовать, когда class пользователя отличается от wheel.

Если модуль дополняет другой модуль, XML-представление данных будет отражать префикс дополняющего модуля. Например, если приведённый выше пример дополнения происходил в модуле с префиксом other, XML будет выглядеть, как показано ниже.

Пример NETCONF XML

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

Оператор augment описан в параграфе 7.15.

4.2.9. Определения RPC

Язык YANG позволяет определять NETCONF RPC. Имена операций, а также входные и выходные параметры моделируются с использованием операторов определения данных YANG.

Пример YANG

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

Пример NETCONF XML

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

Оператор rpc описан в параграфе 7.13.

4.2.10. Определения уведомлений

Язык YANG позволяет определять уведомления, подходящие для NETCONF. Для задания содержимого уведомления применяются операторы определения данных YANG.

Пример YANG

```
notification link-failure {
    description "Был обнаружен отказ канала";
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf if-admin-status {
        type admin-status;
    }
    leaf if-oper-status {
        type oper-status;
    }
}
```

Пример NETCONF XML

```
<notification
    xmlns="urn:ietf:params:netconf:capability:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/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>
```

Оператор notification описан в параграфе 7.14.

5. Концепции языка

5.1. Модули и субмодули

Модуль является базовым блоком определений в YANG. Модуль определяет одну модель данных. Модуль может определять полную, согласованную модель, а также дополнять существующую модель новыми узлами.

Субмодулями называются частичные модули, вносящие вклад в определения модуля в целом. Модуль может включать произвольное число субмодулей, но каждый субмодуль может относиться лишь к одному модулю.

Имена всех стандартных модулей и субмодулей **должны** быть уникальными. Разработчикам модулей и субмодулей YANG **рекомендуется** выбирать имена для своих модулей с учётом возможных конфликтов со стандартными или другими фирменными (enterprise) модулями, например, используя название организации в качестве префикса имени модуля.

Модули используют оператор include для перечисления своих субмодулей и оператор import для ссылок на внешние модули. Аналогично, субмодули используют оператор import для ссылки на другие модули и include для ссылки на субмодули внутри модуля. В модуль или субмодуль **недопустимо** включать субмодули из других модулей, а субмодулям **недопустимо** импортировать свой модуль.

Операторы import и include служат для обеспечения доступности определений из других модулей и субмодулей:

- для того, чтобы модуль или субмодуль мог ссылаться на определения из внешнего модуля, этот внешний модуль **должен** быть импортирован (import).
- для того, чтобы модуль мог ссылаться на определения одного из своих субмодулей, модуль **должен** включать (include) этот субмодуль;
- для того, чтобы субмодуль мог ссылаться на определения из другого субмодуля в том же модуле, первый субмодуль **должен** включать (include) второй субмодуль.

Недопустимо создавать замкнутые цепочки импорта или включения. Например, если модуль a включает субмодуль b, то b не может включать a.

При ссылке на определение из внешнего модуля **должен** применяться локально заданный префикс, за которым следует разделитель в форме двоеточия (:) и внешний идентификатор. Ссылки на определения в локальном модуле также **могут** использовать префиксы. Поскольку встроенные типы данных не относятся к какому-либо модулю и не имеют префикса, для ссылок на встроенные типы данных (например, int32) префиксы применяться не могут.

5.1.1. Импорт и включение по номеру выпуска (Revision)

Опубликованные модули могут независимо изменяться время от времени. Для обеспечения возможности совершенствования модули могут импортироваться с указанием конкретного выпуска (revision). При создании модуля в нем используются текущие выпуски других модулей, которые существовали на момент разработки данного модуля. По мере публикации новых выпусков импортируемых модулей импортирующий модуль не будет автоматически меняться. Если автор модуля готов перейти к более новому выпуску импортируемого модуля, он публикует свой модуль заново с обновлением операторов import. Таким образом авторы модулей явно указывают совместимость с новым выпуском импортируемого модуля.

Для субмодулей ситуация похожа, но проще. Модуль или субмодуль, включающий другие субмодули, должен указать выпуск (revision) включаемых субмодулей. Если субмодуль изменяется, все включающие его модули и субмодули нужно будет изменить, указав ссылку на новый выпуск.

Например, модуль b импортирует модуль a.

```
module a {
    revision 2008-01-01 { ... }
    grouping a {
        leaf eh { .... }
    }
}

module b {
    import a {
        prefix p;
        revision-date 2008-01-01;
    }

    container bee {
        uses p:a;
    }
}
```

Когда автор модуля a публикует новый выпуск, изменения могут оказаться не приемлемыми с точки зрения автора модуля b. Если же новый выпуск пригоден для импорта, автор модуля b публикует свой модуль заново с обновлённым оператором import.

5.1.2. Иерархии модулей

YANG позволяет моделировать данные с множеством иерархий, когда данные могут иметь более одного узла верхнего уровня. Модели с множеством узлов верхнего уровня иногда очень удобны и поддерживаются YANG.

Протокол NETCONF способен передавать любые данные XML в качестве содержимого элементов <config> и <data>. Узлы верхнего уровня модулей YANG представляются внутри этих элементов в произвольном порядке, как дочерние

элементы. Такая инкапсуляция гарантирует, что соответствующие сообщения NETCONF всегда будут корректно сформированными документами XML.

Например, экземпляр

```
module my-config {
    namespace "http://example.com/schema/config";
    prefix "co";

    container system { ... }
    container routing { ... }
}
```

может быть представлен в NETCONF в форме

```
<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="http://example.com/schema/config">
        <!-- system data here -->
      </system>
      <routing xmlns="http://example.com/schema/config">
        <!-- routing data here -->
      </routing>
    </config>
  </edit-config>
</rpc>
```

5.2. Макет файла

Модули и субмодули YANG обычно сохраняются в файлах с одним оператором module или submodule в каждом файле. Для имён файлов следует использовать приведённую ниже форму.

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

Компиляторы YANG могут находить импортируемые модули и включаемые субмодули на базе этих соглашений. Хотя язык YANG определяет модули, инструменты могут компилировать субмодули независимо из соображений производительности и управляемости. Ошибки и предупреждения, которые не могут быть обнаружены во время компиляции субмодуля, могут задерживаться, пока субмодули не будут скомпонованы в единый модуль.

5.3. Пространства имён XML

Все определения YANG задаются внутри модуля, привязанного к определённому пространству имён XML [XML-NAMES], которое является уникальным в глобальном масштабе идентификатором URI [RFC3986]. Клиент и сервер NETCONF используют пространство имён при кодировании данных XML.

Пространства имён XML для модулей, опубликованные в серии RFC [RFC4844], **должны** выделяться агентством IANA (см раздел 14).

Пространства имён XML для приватных модулей выделяются организацией, владеющей модулем, без централизованной регистрации. Идентификаторы URI для пространств имён **должны** выбираться так, чтобы не возникало конфликтов со стандартными или другими фирменными пространствами имён. Например, они могут включать имя компании или организации.

Оператор namespace описан в параграфе 7.1.3.

5.3.1. Пространство имён YANG XML

YANG определяет пространство имён XML для операций NETCONF <edit-config> и содержимого <error-info>. Это пространство имеет имя urn:ietf:params:xml:ns:yang:1.

5.4. Преобразование имён группировок, типов и отождествлений

Имена группировок (grouping), типов (type) и отождествлений (identity) преобразуются в контексте их определения, а не в контексте применения. Пользователи grouping, typedef и identity не обязаны импортировать модули или включать субмодули для соблюдения всех ссылок, заданных исходным определением. Это похоже на область действия (scope) в традиционных языках программирования.

Например, если модуль определяет группировку, в которой присутствует ссылка на тип, при использовании этой группировки во втором модуле тип преобразуется (resolve) в контексте исходного (а не второго) модуля. Благодаря этому не возникает неоднозначностей при определении типа в обоих модулях.

5.5. Вложенные определения типов и группировки

Определения типов (typedef) и группировки (grouping) во многих операторах YANG могут быть вложенными, что позволяет лексически охватывать их иерархией операторов, в которой они появляются. Это даёт возможность определять типы и группировки вблизи места их применения, а не на вершине иерархии. Такая близость улучшает читаемость определений.

Задание области действия также позволяет определять типы без учёта конфликтов имён в разных субмодулях. Имена типов можно задавать без добавления в начале строк, предназначенных для предотвращения конфликтов имён в больших модулях.

Наконец, обозначение области действия позволяет автору модуля сохранить типы и группировки (приватными) внутри модуля или субмодуля и предотвратить их повторное использование (reuse). Поскольку за пределами модуля или субмодуля могут применяться лишь типы и группировки верхнего уровня (т. е. те, которые указаны в субоператорах операторов module или submodule), разработчик получает больше контроля над компонентами своих модулей, представленными во внешний мир, может скрыть внутреннюю структуру и установить границу между видимыми извне и приватными частями.

Определениям области действия **недопустимо** затенять определения с более широкой областью действия. Тип и группировка не могут быть определены, если более высокий уровень в иерархии операторов уже включает определение с таким идентификатором.

Ссылка на тип или группировку без префикса или с префиксом текущего модуля преобразуется (resolve) путём нахождения соответствующего оператора typedef или grouping среди непосредственных субоператоров каждого оператора предка.

5.6. Соответствие

Соответствие модели является мерой того, насколько точно устройство следует модели. Вообще говоря, устройства отвечают за правильную реализацию модели, что позволяет приложениям считать, что устройства одинаково реализуют модель. Отклонения могут сузить возможности модели и увеличить число проблем для использующих её приложений.

Для YANG возможны три варианта соответствия:

- базовое поведение модели;
- необязательные функции в составе модели;
- отклонения от модели.

Эти варианты более подробно рассматриваются ниже.

5.6.1. Базовое поведение

Модель определяет соглашение между клиентом и сервером NETCONF, которое позволяет обеим сторонам верить в то, что другая сторона знает синтаксис и семантику моделируемых данных. Сила YANG заключается в выполнении этого соглашения.

5.6.2. Дополнительные возможности

Во многих моделях авторы делают некоторые части модели не обязательными (условными). Устройство самостоятельно определяет какие из этих частей пригодны и поддерживаются им.

Например, модель данных syslog может включать возможность локального сохранения системных журналов, но разработчик модели понимает, что это возможно лишь при наличии локального хранилища. Если такого хранилища нет, приложению не следует запрашивать у устройства хранение системных журналов.

YANG поддерживает такой механизм условной поддержки с помощью конструкции feature. Это даёт разработчику модели механизм, позволяющий сделать отдельные части модели условными с решением вопроса их поддержки на устройстве. Модель может содержать конструкции, которые поддерживаются не всеми устройствами. Эти функции (возможности) включаются в определение модели, обеспечивая её цельное представление и позволяя приложениям узнать, какие функции поддерживаются для учёта этого в своём поведении.

Модуль может заявлять любое число функций (feature), указываемых простыми строками, и может делать соответствующие части модуля необязательными. Если устройство поддерживает эту функцию, соответствующая часть модуля становится действительной для этого устройства. Если же функция не поддерживается, эта часть модуля становится не пригодной для применения и приложения учитывают это в своём поведении.

Возможности определяются с помощью оператора feature. Определения в модуле, которые являются условными по отношению к этой функции, помечаются оператором if-feature с именем функции (feature) в качестве аргумента.

Дополнительная информация о необязательных функциях (возможностях) приведена в параграфе 7.18.1.

5.6.3. Отклонения от модели

В идеальном мире от всех устройств требуется точная реализация модели и отклонения от неё не разрешаются. Однако в реальности устройства зачастую просто не способны или не предназначены для полной реализации модели. В автоматизации на базе YANG требуются механизмы, с помощью которых устройства смогут информировать приложения о своих отклонениях от модели.

Например, модуль BGP может поддерживать любое число партнёров BGP, а конкретный сервер ограничивается поддержкой 16 партнёров. Любое приложение, пытающееся настроить семнадцатого партнёра, столкнётся с ошибкой. Хотя этой ошибки приложению может оказаться достаточно для того, чтобы понять ограничение, лучше было бы приложению заранее знать об этом и принять соответствующие меры, избавив пользователя от ненужной работы.

Отклонения устройств от модели могут объявляться с использованием оператора deviation, который принимает в качестве аргумента строку с идентификатором узла в дереве схемы. Содержимое этого оператора уточняет информацию об отклонении сервера от заданного в модуле поведения.

Описание оператора deviation приведено в параграфе 7.18.3.

5.6.4. Анонсирование информации о соответствии в сообщении <hello>

Пространство имён URI должно анонсироваться как возможность в сообщении NETCONF <hello> для индикации поддержки модуля YANG сервером NETCONF. URI анонсируемых возможностей должны иметь форму, показанную ниже.

```
     capability-string   = namespace-uri [ parameter-list ]
     parameter-list      = "?" parameter *( "&" parameter )
     parameter           = revision-parameter /
                           module-parameter /
                           feature-parameter /
                           deviation-parameter
     revision-parameter  = "revision=" revision-date
     module-parameter    = "module=" module-name
     feature-parameter   = "features=" feature *( "," feature )
     deviation-parameter = "deviations=" deviation *( "," deviation )
```

Параметр revision-date указывает выпуск модуля (см. параграф 7.1.9), который реализует сервер NETCONF, module-name - имя модуля, указанное в операторе module (см. параграф 7.1), namespace-uri - пространство имён URI для модуля, указанное в операторе namespace (см. параграф 7.1.3), feature - имя необязательной функции, реализованной устройством (см. параграф 7.18.1), а deviation - имя модуля, определяющего отклонения устройства (см. параграф 7.18.3).

В списке параметров каждый поименованный параметр **должен** появляться не более одного раза.

5.6.4.1. Модули

Сервер указывает имена поддерживаемых модулей в сообщении <hello>. Пространства имён модулей кодируются в форме базовых URI в строке возможностей, а имя модуля кодируется в параметре module для базового URI.

Сервер **должен** анонсировать все возможности всех реализуемых им модулей.

Приведённое ниже сообщение <hello> анонсирует единственный модуль syslog.

```
   <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <capability>
       http://example.com/syslog?module=syslog&amp;revision=2008-04-01
     </capability>
   </hello>
```

5.6.4.2. Возможности

Сервер указывает имена поддерживаемых возможностей в сообщении <hello>. В этих сообщениях возможности кодируются в параметре features внутри URI. Значение параметра представляет собой список разделённых запятыми имён возможностей, которые устройство поддерживает для заданного модуля.

Н ниже приведён пример сообщение <hello>, анонсирующего один модуль и информирующего клиента о поддержке возможности local-storage (локальное хранилище) модуля syslog.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capability>
    http://example.com/syslog?module=syslog&amp;features=local-storage
  </capability>
</hello>
```

5.6.4.3. Отклонения

Отклонения устройства анонсируются с помощью параметра deviations, значением которого служит список разделённых запятыми имён модулей, содержащих отклонения от возможностей.

Приведённое ниже сообщение <hello> анонсирует два модуля, информируя клиента о том, что имеются отклонения от возможностей модуля syslog, перечисленные в модуле my-devs.

```
   <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <capability>
         http://example.com/syslog?module=syslog&amp;deviations=my-devs
       </capability>
       <capability>
         http://example.com/my-deviations?module=my-devs
       </capability>
     </hello>
```

5.7. Изменение хранилища данных

Модели данных могут разрешать серверу менять хранилище конфигурации способами, которые не заданы явно в сообщениях протокола NETCONF. Например, модель данных может определять листья, которым присваиваются определённые системой значения, если клиент их не предоставил. Формальный механизм задания обстоятельств, при которых такие изменения разрешены, выходит за рамки этой спецификации.

6. Синтаксис YANG

Синтаксис YANG похож на применяемый в SMIng [RFC3780] и языках программирования типа C или C++. Синтаксис в стиле C был осознанно выбран для удобочитаемости, поскольку YANG считает более важным фактором время и усилия читателей моделей YANG, нежели разработчиков модулей и инструментальных средств. Этот раздел посвящён описанию синтаксиса YANG.

Модули YANG используют кодировку символов UTF-8 [RFC3629].

6.1. Лексические маркеры

Модули YANG разбираются как последовательности маркеров. В этом параграфе рассмотрены правила распознавания маркеров во входном потоке. Правила разбора маркеров YANG просты и обеспечивают широкие возможности. Простота вызвана необходимостью упростить реализацию анализаторов, а мощный набор возможностей нужен разработчикам модулей для того, чтобы выразить модули в читаемой форме.

6.1.1. Комментарии

Для комментариев применяется стиль C++. Одиночная строка комментария начинается символами // и заканчивается как все строки. Комментарий блока строк начинается с последовательности /* и завершается последовательностью */.

6.1.2. Маркеры

Маркером (token) в YANG считается ключевое слово, строка, точка с запятой (;) и фигурные скобки ({ и }). Строка может быть заключена в кавычки. Ключевое слово - это одно из ключевых слов YANG, определённых в этом документе, или идентификатор префикса, за которым следует двоеточие (:) и ключевое слово расширения языка. Регистр символов в ключевых словах принимается во внимание. Формальное определение идентификаторов дано в параграфе 6.2.

6.1.3. Кавычки

Если строка содержит пробелы, символы табуляции, точку с запятой (;), скобки ({ или }) или последовательности для обозначения комментариев (//, /*, */), она **должна** заключаться в двойные или одинарные кавычки.

Если строка в двойных кавычках включает символы завершения строки, за которыми следуют пробелы или табуляторы, используемые для задания отступа в соответствии с макетом файла YANG, эти пробельные символы в начале строк вырезаются вплоть до столбца с открывающей двойной кавычкой (включая его) или первого непробельного символа (что встретится раньше). Любой символ табуляции перед проверкой преобразуется в 8 символов пробела.

Если строка в двойных кавычках содержит символы табуляции или пробела перед завершением строки, эти символы вырезаются.

Строка в одинарных кавычках (' ') сохраняет значение каждого символа. Символ одинарных кавычек не может использоваться в таких строках даже вместе с символом \.

Внутри строк в двойных кавычках (" ") символ \ имеет специальное значение, которое зависит от следующего непосредственно за ним символа:

\n новая строка;

\t символ табуляции;

\" символ двойной кавычки;

\\ одиночный символ \.

Если после строки в кавычках следует символ сложения (+) и другая строка в кавычках, эти строки объединяются в одну (конкатенация), что позволяет собирать одну большую строку из множества строк ограниченной длины. Перед конкатенацией строк выполняется удаление пробельных символов в начале и замена последовательностей специального назначения с \.

6.1.3.1. Примеры кавычек

Ниже приведён набор эквивалентных строк.

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

Далее показаны несколько строк специального назначения.

```
"\""  - строка, содержащая двойную кавычку
'"'   - строка, содержащая двойную кавычку
"\n"  - строка, содержащая символы новой строки
'\n'  - строка, содержащая символ \, за которым следует символ n
```

Ниже приведены два примера недопустимых строк.

```
''''  - строка в одинарных кавычках не может включать одинарные кавычки
"""   - символ двойных кавычек должен использовать префикс \
```

Две приведённых ниже строки эквивалентны.

```
    "first line
       second line"

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

6.2. Идентификаторы

Идентификаторы служат для обозначения различных элементов YANG по именам. Каждый идентификатор начинается с буквы кода ASCII (в верхнем или нижнем регистре) или символа подчёркивания, а далее могут следовать дополнительные буквы, цифры, символы подчёркивания, дефиса и точки в кодировке ASCII. Реализации **должны** поддерживать идентификаторы размером до 64 символов. Регистр символов в идентификаторах принимается во внимание. Синтаксис идентификаторов формально определён правилом identifier в разделе 12. Идентификаторы могут задаваться в форме строк в кавычках и без кавычек.

6.2.1. Идентификаторы и их пространства имён

Каждый идентификатор действителен в пространстве имён, которое зависит от типа определяемого элемента YANG. Все идентификаторы в одном пространстве имён **должны** быть уникальными.

- Все имена модулей и субмодулей используют общее глобальное пространство имён модулей.
- Все имена расширений, определённых в модуле и его субмодулях, используют общее пространство имён.

- Все имена функций (feature), определённых в модуле и его субмодулях, используют одно пространство имён.
- Все имена отождествлений (identity) определённых в модуле и его субмодулях, используют общее пространство имён отождествлений.
- Все имена производных типов, определённых в родительском узле или на верхнем уровне модуля и его субмодулей, используют общее пространство имён идентификаторов типа. Это пространство имён охватывает все дочерние узлы модуля или родительского узла. Это означает, что любой узел-потомок может использовать данное определение типа (typedef), а использование typedef с таким же именем **недопустимо**.
- Все имена группировок, определённых в родительском узле или на верхнем уровне модуля и его субмодулей, используют общее пространство имён группировок. Это пространство имён охватывает все дочерние узлы модуля или родительского узла. Это означает, что любой узел-потомок может использовать данное группировку (grouping), а использование этого же имени для других группировок **недопустимо**.
- Все узлы типа leaf, leaf-list, list, container, choice, rpc, notification и anyxml, определённые (напрямую или с помощью оператора uses) в родительском узле или на верхнем уровне модуля и его субмодулей определённых в модуле и его субмодулях, используют общее пространство имён. Это пространство охватывает родительский узел или модуль, если родительский узел не относится к типу case. В последнем случае пространство имён охватывает ближайшего предка, не являющегося узлом типа case или choice.
- Все узлы case внутри choice используют общее пространство имён идентификаторов. Это пространство охватывает родительский узел choice.

В YANG разрешены ссылки вперёд (упреждающие).

6.3. Операторы

Модуль YANG содержит последовательность операторов. Каждый оператор начинается с ключевого слова, за которым могут следовать аргументы, а затем символ ; или блок субоператоров, заключенный в фигурные скобки ({ })

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

Аргумент является строкой (см. параграф 6.1.2).

6.3.1. Расширения языка

Модуль может задавать расширения языка YANG с помощью ключевого слова extension (см. параграф 7.17). Расширения могут импортироваться другими модулями с помощью оператора import (см. параграф 7.1.5). При использовании импортированного расширения ключевое слово расширения **должно** указываться с префиксом, который применялся для импорта модуля расширения. Если расширение применяется в определившем его модуле, ключевое слово расширения **должно** указываться с префиксом этого модуля.

Поскольку субмодули не могут включать родительский модуль, любые расширения в модуле, которые нужно показать субмодулям, **должны** быть определены в субмодуле. Субмодули тогда смогут включать этот субмодуль, чтобы найти определение расширения.

Если компилятор YANG не поддерживает того или иного расширения, которое присутствует в модуле YANG как неизвестный оператор (см. раздел 12), такой оператор **можно** игнорировать целиком.

6.4. Вычисление XPath

Язык YANG опирается на язык путей XML (XPath[1]) 1.0 [XPATH] в качестве нотации для задания множества связей и зависимостей между узлами. Клиенты и серверы NETCONF не обязаны включать интерпретатор XPath, но они **должны** гарантировать выполнение требований, представленных в модели данных. Способ исполнения этого зависит от реализации. Выражения XPath **должны** быть корректны синтаксически, а все используемые префиксы **должны** присутствовать в контексте XPath (см. параграф 6.4.1). Реализация может сделать это вручную вместо использования выражения XPath напрямую.

Модель данных, применяемая в выражении XPath, совпадает с используемой в XPath 1.0 [XPATH] с тем же расширением для потомка корневого узла, что применяется в XSLT 1.0 (см. параграф 3.1 в [XSLT]). Это означает, в частности, что корневой узел может иметь любое число элементов в качестве своих потомков.

6.4.1. Контекст XPath

Все выражения YANG XPath используют приведённое ниже определение контекста XPath.

- Набор деклараций пространства имён представляет собой набор всех пар префиксов операторов import и пространств имён в модуле, где задано выражение XPath, и всех префиксов операторов prefix для пространства имён URI оператора.
- Имена без префикса пространства имён относятся к тому же пространству, что идентификатор текущего узла. Внутри группировки это пространство зависит от того, где группировка используется (см. параграф 7.12).
- Библиотека функций - это основная библиотека, определённая в [XPATH], и функция current(), которая возвращает набор узлов с начальным узлом контекста.
- Набор привязок переменных пуст.

Механизм обработки имён без префиксов приспособлен из XPath 2.0 [XPATH2.0] и помогает упростить выражения XPath в YANG. Неоднозначностей возникать не может, поскольку идентификаторы узлов YANG всегда являются квалифицированными именами с непустым URI пространства имён.

Узел контекста меняется в зависимости от выражения YANG XPath и указывается там, где определён оператор YANG с выражением XPath.

[1]XML Path Language - язык путей XML.

6.5. Идентификатор узла схемы

Идентификатор узла схемы представляет собой строку, указывающую узел в дереве схемы. Он имеет две формы - абсолютную и наследуемую (descendant), которые определены правилами absolute-schema-nodeid и descendant-schema-nodeid в разделе 12. Идентификатор узла схемы состоит из пути идентификаторов, разделённых символами дробной черты (/). В абсолютном идентификаторе первым символом является /, а за ним следует узел схемы верхнего уровня в локальном или всех импортированных модулях.

Ссылки на идентификаторы, определённые во внешних модулях, **должны** включать соответствующие префиксы, а ссылки на идентификаторы, определённые в текущем модуле и его субмодулях, **могут** использовать префикс.

Например, для указания дочернего узла b узла верхнего уровня a может использоваться строка /a/b.

7. Операторы YANG

В этом разделе описаны все операторы YANG.

Отметим, что даже операторы, не имеющие каких-либо субоператоров, определённых в YANG, могут иметь в качестве субоператоров фирменные расширения. Например, оператор description не имеет расширений в YANG, но возможен показанный ниже вариант субоператора.

```
description "Некий текст" {
    acme:documentation-flag 5;
}
```

7.1. Оператор module

Оператор module определяет имя модуля и собирает воедино все относящиеся к модулю операторы. Аргументом оператора module является имя модуля, за которым следует блок субоператоров с дополнительной информацией о модуле. Имя модуля следует правилам для идентификаторов (см. параграф 6.2).

Имена модулей, публикуемые в RFC [RFC4844], **должны** выделяться IANA (см. раздел 14).

Частные (фирменные) имена модулей назначаются владеющими модулями организациями без централизованной регистрации. **Рекомендуется** выбирать имена модулей с учётом возможных конфликтов с другими фирменными модулями и субмодулями. Например, можно использовать название предприятия или организации в качестве префикса для имени модуля.

Модуль обычно имеет показанную ниже структуру.

```
module <module-name> {

    // информация заголовка
    <оператор yang-version>
    <оператор namespace>
    <оператор prefix>

    // данные о привязках
    <операторы import>
    <операторы include>

    // метаданные
    <оператор organization>
    <оператор contact>
    <оператор description>
    <оператор reference>

    // история выпусков
    <операторы revision>

    // определения модуля
    <другие операторы>
}
```

7.1.1. Субоператоры для module

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
augment	7.15	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.19.3	0..1
deviation	7.18.3	0..n
extension	7.17	0..n
feature	7.18.1	0..n
grouping	7.11	0..n
identity	7.16	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	7.19.4	0..1
revision	7.1.9	0..n
rpc	7.13	0..n
typedef	7.3	0..n
uses	7.12	0..n
yang-version	7.1.2	1

7.1.2. Оператор yang-version

Необязательный оператор yang-version указывает версию языка YANG, использованную при разработке модуля. Аргументом оператора является строка. Для модулей YANG, определённых на основе данной спецификации, она **должна** иметь значение1.

Обработка операторов yang-version со значением версии, отличающимся от 1 (определена здесь), выходит за рамки данной спецификации. Любой документ, определяющий более высокий номер версии должен определять совместимость этой версии с более ранними.

7.1.3. Оператор namespace

Оператор namespace определяет пространство имён XML, в котором все определённые модулем идентификаторы представляются в XML (за исключением идентификаторов узлов данных внутри группировок - см. параграф 7.13). Аргументом оператора namespace служит идентификатор URI пространства имён.

См. также параграф 5.3.

7.1.4. Оператор prefix

Оператор prefix служит для определения префикса, связанного с модулем и его пространством имён. Аргументом оператора prefix является строка префикса, которая применяется для доступа к модулю. Строка префикса может использоваться в модуле для ссылок на содержащиеся в нем определения (например, if:ifName). Для префиксов применяются такие же правила, как для идентификаторов (см. параграф 6.2).

При использовании в операторе module субоператор prefix определяет префикс, предлагаемый для использования при импорте этого модуля. Для удобочитаемости NETCONF XML клиенту или серверу NETCONF, генерирующему XML или XPath с использованием префиксов, следует применять определённый в модулей префикс в качестве префикса пространства имён XML, если это не вызывает конфликтов.

При использовании в операторе import субоператор prefix определяет префикс, используемый для доступа к определениям в импортируемом модуле. При использовании ссылок на операторы из импортируемого модуля за строкой префикса этого модуля следует двоеточие (:) и идентификатор (например if:ifIndex). Для удобочитаемости модулей YANG определённый модулем префикс **следует** использовать при импорте модуля, если это не вызывает конфликтов. При возникновении конфликта когда импортируются два разных модуля, имеющих одинаковые префиксы, хотя бы один из них **должен** импортироваться с другим префиксом.

Все префиксы, включая и префикс самого модуля, **должны** быть уникальными в рамках модуля или субмодуля.

7.1.5. Оператор import

Оператор import делает доступными операторы из другого модуля или субмодуля. Аргументом оператора служит имя импортируемого модуля, за которым следует блок субоператоров с данными импорта. Импортирующий модуль может:

- использовать любые группировки (grouping) и определения типов (typedef), заданные на верхнем уровне импортированного модуля и его субмодулей;
- применять любые расширения, функции и отождествления из импортированного из модуля и его субмодулей;
- использовать любой узел из дерева схемы импортированного модуля в операторах must, path и when или в качестве целевого узла в операторах augment и deviation.

Обязательный субоператор prefix задаёт префикс для импортируемого модуля в области действия импортирующего модуля и его субмодулей. Для импорта разных модулей можно использовать множество операторов import.

При наличии необязательного субоператора revision-date любые typedef, grouping, extension, feature и identity, указанные в определениях локального модуля, берутся из заданного выпуска импортируемого модуля. Если указанного выпуска импортируемого модуля не существует, возникает ошибка. При отсутствии субоператора revision-date выпуск импортируемого модуля будет не известен.

Недопустимо импортировать множество выпусков одного модуля.

Субоператор	*Параграф*	*Число элементов*
prefix	7.1.4	1
revision-date	7.1.5.1	0..1

7.1.5.1. Оператор revision-date для оператора import

Субоператор revision-date используется в операторе import для указания выпуска импортируемого модуля. Оператор revision-date **должен** соответствовать наиболее свежему оператору revision в импортируемом модуля.

7.1.6. Оператор include

Оператор include делает содержимое субмодуля доступным в родительском модуле или другом субмодуле родительского модуля. Аргументом оператора служит имя включаемого субмодуля. В модули можно включать лишь относящиеся к ним субмодули, указанные оператором belongs-to (см. параграф 7.2.2). В субмодули можно включать лишь другие субмодули того же модуля.

Когда модуль включает в себя субмодуль, он встраивает содержимое субмодуля в свою иерархию узлов. Когда субмодуль включает в себя другой субмодуль, определения включаемого субмодуля становятся доступными ему.

При наличии необязательного субоператора revision-date будет включаться указанный выпуск субмодуля. Если указанного выпуска не существует, возникает ошибка. Если субоператор revision-date не задан, выпуск субмодуля будет не известен.

Недопустимо включение в модуль множества выпусков одного субмодуля.

Субоператор	*Параграф*	*Число элементов*
revision-date	7.1.5.1	0..1

7.1.7. Оператор organization

Оператор organization указывает сторону, ответственную за этот модуль. Аргументом оператора является строка с текстовым описанием организации, под эгидой которой был разработан модуль.

7.1.8. Оператор contact

Оператор contact указывает контактные сведения для модуля. Аргументом оператора является строка с контактными данными ответственных за техническое обслуживание модуля, включая имя, почтовый адрес, номер телефона и адрес электронной почты.

7.1.9. Оператор revision

Оператор revision указывает историю изменения модуля, включая его начальный выпуск. Последовательность операторов revision детализирует изменения в определении модуля. Аргументом оператора служит строка даты в формате YYYY-MM-DD (год-месяц-число), за которой следует блок субоператоров с информацией о выпуске. Модулю **следует** включать по крайней мере один оператор revision. Для каждого опубликованного выпуска **следует** добавлять новый оператор в начале последовательности, чтобы все выпуски перечислялись в обратном хронологическом порядке.

7.1.9.1. Субоператоры для revision

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
reference	7.19.4	0..1

7.1.10. Пример использования

```
module acme-system {
    namespace "http://acme.example.com/system";
    prefix "acme";

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

    include acme-types;

    organization "ACME Inc.";
    contact
        "Joe L. User

         ACME, Inc.
         42 Anywhere Drive
         Nowhere, CA 95134
         USA

         Phone: +1 800 555 0100
         EMail: joe@acme.example.com";

    description
        "Модуль для объектов, реализующих протокол ACME.";

    revision "2007-06-09" {
        description "Первый выпуск.";
    }

    // определения ...
}
```

7.2. Оператор submodule

Хотя основным блоком языка YANG является модуль, модули YANG сами могут включать субмодули. Это позволяет разделить сложную модель на несколько частей, где субмодули используют общее пространство имён, определённое родительским модулем.

Оператор submodule задаёт имя субмодуля и группирует все операторы, относящиеся к этому субмодулю. Аргументом оператора submodule является имя субмодуля, за которым следует блок операторов с информацией о субмодуле. Имя субмодуля является идентификатором (см. параграф 6.2).

Имена субмодулей, публикуемые в RFC [RFC4844], **должны** выделяться IANA (см. раздел 14).

Частные (фирменные) имена субмодулей назначаются владеющими модулями организациями без централизованной регистрации. **Рекомендуется** выбирать имена для субмодулей с учётом возможных конфликтов со стандартными и

другими фирменными модулями и субмодулями. Например, можно использовать в качестве префикса имени субмодуля название предприятия или организации.

Субмодуль обычно имеет показанную ниже структуру.

```
submodule <module-name> {
    <оператор yang-version>
    // идентификация модуля
    <оператор belongs-to>
    // операторы привязки
    <операторы import>
    <операторы include>
    // метаданные
    <оператор organization>
    <оператор contact>
    <оператор description>
    <оператор reference>
    // история выпусков
    <операторы revision>
    // определения субмодуля
    <другие операторы>
}
```

7.2.1. Субоператоры для submodule

Субоператор	Параграф	Число элементов
anyxml	7.10	0..n
augment	7.15	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	7.19.3	0..1
deviation	7.18.3	0..n
extension	7.17	0..n
feature	7.18.1	0..n
grouping	7.11	0..n
identity	7.16	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	7.14	0..n
organization	7.1.7	0..1
reference	7.19.4	0..1
revision	7.1.9	0..n
rpc	7.13	0..n
typedef	7.3	0..n
uses	7.12	0..n
yang-version	7.1.2	0..1

7.2.2. Оператор belongs-to

Оператор belongs-to указывает модуль, к которому субмодуль относится. Аргументом оператора является имя модуля.

Субмодуль **должен** включаться только в модуль, к которому он относится или в его субмодули.

Обязательный субоператор prefix указывает префикс для модуля, к которому относится этот субмодуль. Все определения в этом модуле и всех его субмодулях доступны при использовании этого префикса.

Субоператор	Параграф	Число элементов
prefix	7.1.4	1

7.2.3. Примеры использования

```
submodule acme-types {

    belongs-to "acme-system" {
        prefix "acme";
    }

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

    organization "ACME Inc.";
    contact
        "Joe L. User

         ACME, Inc.
         42 Anywhere Drive
         Nowhere, CA 95134
         USA
```

```
            Phone: +1 800 555 0100
            EMail: joe@acme.example.com";

        description
            "Этот субмодуль определяет общие типы ACME.";

        revision "2007-06-09" {
            description "Первый выпуск.";
        }

        // далее следуют определения ...
    }
```

7.3. Оператор typedef

Оператор typedef определяет новый тип, который может применяться локально в модуле или субмодуле, а также в других модулях, импортирующих данный, в соответствии с правилами, описанными в параграфе 5.5. Новый тип называется производным (derived), а тип, на основе которого он создан, - базовым. Все производные типы можно отследить по цепочке до встроенного типа YANG.

Аргументом оператора typedef является идентификатор, задающий имя определяемого типа, за которым **должен** следовать блок субоператоров с подробной информацией об определяемом типе.

В качестве имени производного типа **недопустимо** использовать какое-либо из имён встроенных типов YANG. Если typedef определяется на верхнем уровне модуля или субмодуля YANG, имя определяемого типа **должно** быть уникальным в данном модуле.

7.3.1. Субоператоры для typedef

Субоператор	*Параграф*	*Число элементов*
default	7.3.4	0..1
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
type	7.3.2	1
units	7.3.3	0..1

7.3.2. Оператор type

Оператор type, который **должен** присутствовать, определяет базовый тип, на основе которого создаётся производный (см. параграф 7.4).

7.3.3. Оператор units

Необязательный оператор units, принимает в качестве аргумента строку, которая даёт текстовое определение единиц, связанных с типом.

7.3.4. Оператор default

Оператор default принимает аргумент в виде строки с принятым по умолчанию значением для нового типа.

Заданное оператором default значение **должно** быть пригодным для типа, указанного в операторе type.

Если базовый тип имеет принятое по умолчанию значение, а новый производный тип такого значения не задаёт, принятое по умолчанию значение базового типа будет наследоваться производным типом.

Если заданное в качестве принятого по умолчанию значение не пригодно по причине новых ограничений, заданных для производного типа или определения листа, определение производного типа или листа **должно** указать новое значение, совместимое с этими ограничениями.

7.3.5. Пример использования

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

7.4. Оператор type

Оператор type принимает один аргумент в форме строки с именем встроенного типа YANG (см. раздел 9) или производного типа (см. параграф 7.3), за которым может следовать блок субоператоров, используемых для дополнительных ограничений типа.

Ограничения, которые могут быть применены, зависят от типа. Операторы ограничений для всех встроенных типов описаны в параграфах раздела 9.

7.4.1. Субоператоры для type

Субоператор	*Параграф*	*Число элементов*
bit	9.7.4	0..n
enum	9.6.4	0..n
length	9.4.4	0..1
path	9.9.2	0..1
pattern	9.4.6	0..n
range	9.2.4	0..1
require-instance	9.13.2	0..1
type	7.4	0..n

7.5. Оператор container

Оператор container служит для определения внутренних узлов данных в дереве схемы. Он принимает один аргумент, служащий идентификатором, за которым следует блок субоператоров с детальной информацией о контейнере.

Узел-контейнер на имеет значения, но имеет список дочерних узлов дерева данных. Дочерние узлы определяются в субоператорах контейнера.

7.5.1. Контейнеры с presence

YANG поддерживает два типа контейнеров - те, которые существуют только для организации иерархии узлов данных, и те, чьё присутствие в конфигурации имеет явное значение.

В первом случае контейнер не имеет своего значения и существует лишь для организации дочерних узлов. Этот стиль применяется по умолчанию.

Например, набор опций скрэмблирования для интерфейсов SONET[1] может помещаться в контейнер scrambling для более эффективной организации конфигурационной иерархии и совместного хранения этих узлов. Сам узел scrambling не имеет значения (смысла), поэтому удаление узла, когда он становится пустым, освобождает пользователя от выполнения этой задачи.

Во втором варианте наличие контейнера само по себе несёт некую информацию, представляя один бит данных конфигурации. Для конфигурационных данных контейнер выступает в качестве конфигурационного элемента и средства организации связанных конфигурационных узлов. Такие контейнеры создаются и удаляются явно.

YANG называет контейнеры этого типа контейнерами присутствия (presence container) и они обозначаются оператором presence, который принимает в качестве аргумента текстовую строку, указывающую смысл присутствия узла.

Например, контейнер ssh может включать возможность входа на сервер по протоколу SSH[2], но может также включать любые связанные с SSH конфигурационные элементы, такие как скорость соединения или число попыток.

Оператор presence (см. параграф 7.5.5) служит для представления смысла наличия контейнера в дереве данных.

7.5.2. Субоператоры для container

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
config	7.19.1	0..1
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
presence	7.5.5	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
uses	7.12	0..n
when	7.19.5	0..1

7.5.3. Оператор must

Необязательный оператор must принимает один аргумент в виде строки с выражением XPath (см. параграф 6.4). Он используется для формального выражения ограничений на пригодные данные. Ограничение применяется в соответствии с правилами, указанными в разделе 8.

При проверке хранилища данных на пригодность, все ограничения must концептуально проверяются один раз для каждого узла в дереве данных и для всех используемых листьев с принятыми по умолчанию значениями (см. параграф 6.4.1). Если узла нет в дереве данных или лист не имеет принятого по умолчанию значения, оператор must не оценивается.

Чтобы данные считались пригодными проверка всех таких ограничений **должна** давать значение true.

Выражения XPath концептуально оцениваются в описанном ниже контексте, дополняющем определение параграфа 6.4.1.

- Узлом контекста является узел в дереве данных, для которого определён оператор must.
- Доступное дерево состоит из всех узлов в дереве данных и всех листьев с используемыми значениями, которые приняты по умолчанию (параграф 7.6.1).

Доступное дерево зависит от узла контекста.

- Если узел контекста представляет конфигурацию, дерево представляет данные в хранилище NETCONF, где существует узел контекста. Корневой узел XPath имеет в качестве потомков все узлы конфигурационных данных верхнего уровня во всех модулях.
- Если узел контекста представляет данные состояния, деревом являются все данные состояния на устройстве и хранилище <running/>. Корневой узел XPath имеет в качестве потомков все узлы данных во всех модулях.

[1]Synchronous Optical Network - синхронная оптическая сеть.
[2]Secure SHell - защищенная командная оболочка.

- Если узел контекста представляет содержимое уведомления, дерево является документом экземпляра XML для уведомления. Корневой узел XPath имеет в качестве единственного потомка элемент, представляющий уведомление.
- Если узел контекста представляет входные параметры RPC, дерево является документом экземпляра RPC XML. Корневой узел XPath имеет в качестве единственного потомка элемент, представляющий операцию RPC, которая будет определена.
- Если узел контекста представляет выходные параметры RPC, дерево является экземпляром документа отклика RPC. Корневой узел XPath имеет в качестве потомков элементы, представляющие выходные параметры RPC.

Результат выражения XPath преобразуется в логическое значение с использованием стандартных правил XPath.

Поскольку все значения листьев в дереве данных концептуально хранятся в канонической форме (см. параграфы 7.6 и 7.7), любые сравнение XPath также выполняются для канонических значений.

Отметим также, что выражение XPath оценивается концептуально. Это означает, что реализация не использует оценщик XPath на устройстве. Однако на практике оценка осуществляется по усмотрению реализации.

7.5.4. Субоператоры для must

Субоператор	Параграф	Число элементов
description	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	0..1

7.5.4.1. Оператор error-message

Необязательный оператор error-message принимает в качестве аргумента строку. Если оценка ограничений даёт значение false, строка передаётся как <error-message> в <rpc-error>.

7.5.4.2. Оператор error-app-tag

Необязательный оператор error-app-tag принимает в качестве аргумента строку. Если оценка ограничений даёт значение false, строка передаётся как <error-app-tag> в <rpc-error>.

7.5.4.3. Пример использования must и error-message

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

7.5.5. Оператор presence

Оператор presence указывает смысл присутствия оператора container в дереве данных. Он принимает в качестве аргумента строку с текстовым описанием смысла присутствия узла.

Если контейнер имеет оператор presence, наличие контейнера в дереве данных имеет некий самостоятельный смысл. В противном случае контейнер служит для представления некой структуры данных и не имеет самостоятельного значения.

Дополнительная информация приведена в параграфе 7.5.1.

7.5.6. Операторы дочерних узлов контейнера

Внутри контейнеры операторы container, leaf, list, leaf-list, uses, choice и anyxmls могут применяться для определения дочерних узлов контейнера.

7.5.7. Правила отображения XML

Узел container представляется в виде элемента XML. Локальное имя элемента служит идентификатором контейнера, а его пространство имён совпадает с пространством имён XML для модуля (см. параграф 7.1.3).

Дочерние узлы контейнера представляются как субэлементы элемента container. Если контейнер определяет входные или выходные параметры RPC или операции (action), эти субэлементы представляются в том же порядке, в котором они определены в операторе container. В остальных случаях субэлементы могут представляться в любом порядке.

Сервер NETCONF, отвечающий на запрос <get> или <get-config>, **может** отказаться от передачи элемента container, если узел контейнера не имеет оператора presence и дочерних узлов. Поэтому клиент, принявший <rpc-reply> для

запроса <get> или <get-config>, должен быть готов обрабатывать ситуации, когда узел контейнера без оператора presence не присутствует в XML.

7.5.8. Операции NETCONF <edit-config>

Контейнеры могут создаваться, удаляться, заменяться и изменяться с помощью операции <edit-config> с атрибутом operation (см. параграф 7.2 в [RFC4741]) в элементе XML этого контейнера.

Если контейнер не имеет оператора presence и последний дочерний узел удалён, сервер NETCONF **может** удалить этот контейнер.

При обработке сервером NETCONF запроса <edit-config> выполняются следующие правила:

- если задана операция merge или replace, узел создаётся при его отсутствии;
- если задана операция create, узел создаётся при его отсутствии, а в случае наличия узла возвращается ошибка data-exists;
- если задана операция delete, узел удаляется при его наличии, а в случае отсутствия узла возвращается ошибка data-missing.

7.5.9. Пример использования

Для приведённого ниже определения контейнера

```
container system {
    description "Содержит различные параметры системы";
    container services {
        description "Настройка доступных извне служб";
        container "ssh" {
            presence "Включает SSH";
            description "Настройки службы SSH";
            // дополнительные листья, контейнеры и т. п. ...
        }
    }
}
```

Соответствующий экземпляр XML будет иметь вид

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

Присутствие элемента <ssh> разрешает использовать SSH.

Удаление контейнера с помощью <edit-config> имеет вид

```
<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="http://example.com/schema/config">
        <services>
          <ssh nc:operation="delete"/>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.6. Оператор leaf

Оператор leaf служит для определения листа (leaf node) в дереве схемы. Он принимает одни аргумент, являющийся идентификатором, за которым следует блок субоператоров, детализирующих информацию листа.

Узел leaf имеет значение, но не имеет дочерних узлов в дереве данных. Понятно, что значение в дереве данных всегда имеет каноническую форму (см. параграф 9.1).

Узлы типа leaf являются необязательными и могут присутствовать во множестве экземпляров.

Оператор leaf используется для определения скалярных переменных конкретного встроенного или производного типа.

7.6.1. Значение листа по умолчанию

Значением листа по умолчанию является используемое сервером значение при отсутствии этого листа в дереве данных. Использование принятого по умолчанию значения зависит от ближайшего предка в дереве схемы, который не является контейнером без присутствия.

- Если такого предка нет в дереве схемы, **должно** использоваться принятое по умолчанию значение.
- В противном случае, если предок является узлом case, принятое по умолчанию значение **должно** использоваться, когда любой узел из case существует в дереве данных или узел case является используемым по умолчанию для choice и нет узлов от других вариантов case в дереве данных.

- В остальных случаях принятое по умолчанию значение **должно** использоваться при наличии предка в дереве данных.

В этих случаях говорят, что используется принятое по умолчанию значение.

При использовании принятого по умолчанию значения сервер **должен** вести себя так, будто лист присутствует в дереве и имеет значение, совпадающее с принятым по умолчанию.

Если лист включает оператор default, по умолчанию для листа будет применяться значение, указанное в операторе default. В остальных случаях, если тип листа имеет принятое по умолчанию значение и не является обязательным, для этого листа по умолчанию будет использоваться принятое по умолчанию значение для этого типа. Во всех остальных случаях лист просто не имеет принятого по умолчанию значения.

7.6.2. Субоператоры для leaf

Субоператор	*Параграф*	*Число элементов*
config	7.19.1	0..1
default	7.6.4	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.19.5	0..1

7.6.3. Оператор type

Оператор type, который **должен** присутствовать, принимает в качестве аргумента имя существующего встроенного или производного типа. Необязательный блок субоператоров задет ограничения для типа (см. параграф 7.4).

7.6.4. Оператор default

Необязательный оператор default принимает в качестве аргумента строку, задающую принятое по умолчанию значение для листа.

Значение оператора default **должно** соответствовать типу листа, заданному оператором type.

Оператор default **недопустимо** включать для узлов, где в качестве значения mandatory установлено true.

Определение принятого по умолчанию значение **недопустимо** помечать оператором if-feature. Например, ниже представлено неприемлемое определение.

7.6.5. Оператор mandatory

Необязательный оператор принимает в качестве аргумента строку true или false и задаёт ограничения для пригодных данных. Если значение оператора не задано, по умолчанию используется false.

Если для mandatory установлено значение true, поведение ограничений зависит от типа ближайшего предка leaf в дереве схемы, который не является контейнером без присутствия (см. параграф 7.5.1):

- если такого предка нет в дереве схемы, leaf **должен** существовать;
- в противном случае, если предок имеет тип case leaf **должен** существовать при наличии в дереве данных хотя бы одного узла из case;
- в остальных случаях leaf **должен** существовать, если узел-предок существует в дереве данных.

Эти ограничения применяются в соответствии с правилами раздела 8.

7.6.6. Правила отображения XML

Узел leaf представляется в виде элемента XML. Локальное имя элемента является идентификатором листа, а его пространством имён является пространство имён XML для модуля (см. параграф 7.1.3).

Значение узла leaf представляется в XML в соответствии с типом и передаётся в элемент в виде символов.

Сервер NETCONF, отвечающий на запрос <get> или <get-config>, **может** отказаться от передачи элемента leaf, если узел leaf имеет принятое по умолчанию значения. Поэтому клиент, принявший <rpc-reply> для запроса <get> или <get-config>, должен быть готов обрабатывать ситуации, когда узел leaf не присутствует в XML. В таких случаях известно, что сервер воспользовался принятым по умолчанию значением.

Пример представления дан в параграфе 7.6.8.

7.6.7. Операции NETCONF <edit-config>

При обработке сервером NETCONF запроса <edit-config> для узла leaf выполняются следующие правила:

- если задана операция merge или replace, узел создаётся при его отсутствии и для него устанавливается значение, найденное в данных XML RPC;
- если задана операция create, узел создаётся при его отсутствии, а в случае наличия узла возвращается ошибка data-exists;

- если задана операция delete, узел удаляется при его наличии, а в случае отсутствия узла возвращается ошибка data-missing.

7.6.8. Пример использования

Ниже приведён оператор leaf, помещённый в определённый выше контейнер ssh (см. параграф 7.5.9):

```
leaf port {
    type inet:port-number;
    default 22;
    description "Порт, который прослушивает сервер SSH."
}
```

Пример соответствующего экземпляра XML будет иметь вид

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

Для установки значения leaf с помощью <edit-config> служит

```
<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="http://example.com/schema/config">
        <services>
          <ssh>
            <port>2022</port>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.7. Оператор leaf-list

Оператор leaf служит для определения простой скалярной переменной того или иного типа, а оператор leaf-list - для определения массива определённого типа. Оператор leaf-list принимает в качестве параметра идентификатор, за которым следует блок субоператоров с детальной информацией о leaf-list. В данных конфигурации все значения в leaf-list **должны** быть уникальными. Понятно, что значения в дереве данных **должны** иметь каноническую форму (см. параграф 9.1).

Если тип, указанный leaf-list, имеет принятое по умолчанию значение, он не оказывает влияния на leaf-list.

7.7.1. Упорядочение

YANG поддерживает два стиля упорядочения элементов в узлах list и leaf-list. Во многих списках порядок элементов не влияет на реализацию конфигурации списка и устройство может сортировать элементы по своему усмотрению. Строка description для списка может предложить порядок для разработчиков реализации сервера. В языке YANG такой стиль называется системным упорядочением (system ordered), а списки помечаются оператором ordered-by system.

Например, список допустимых пользователей обычно сортируется по алфавиту, поскольку порядок расположения пользователей в конфигурации не влияет на создание учётных записей этих пользователей.

В другом стиле списков порядок имеет значение для реализации конфигурации списка и пользователь отвечает за упорядочение элементов, а устройство этот порядок поддерживает. В YANG такие списки называются упорядоченными пользователем (user ordered) и указываются оператором ordered-by user.

Например, список, в соответствии с которым применяются фильтры пакетов к входящему трафику, может влиять на результат фильтрации. Пользователь должен сам решить следует применять фильтр, отбрасывающий все пакеты TCP, до или после фильтра, разрешающего трафик с доверенных интерфейсов. Выбор порядка может быть важен.

YANG обеспечивает в операции NETCONF <edit-config> многочисленные возможности, позволяющие контролировать порядок элементов в списках упорядочиваемых пользователем. Элементы списка могут вставляться или перемещаться, помещаться в начало или конец списка, а также в определённую позицию перед заданным элементом или после него.

Оператор ordered-by описан в параграфе 7.7.57.

7.7.2. Субоператоры для leaf-list

Субоператор	*Параграф*	*Число элементов*
config	7.19.1	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
max-elements	7.7.4	0..1
min-elements	7.7.3	0..1
must	7.5.3	0..n
ordered-by	7.7.5	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
type	7.4	1
units	7.3.3	0..1
when	7.19.5	0..1

7.7.3. Оператор min-elements

Необязательный оператор min-elements принимает в качестве аргумента неотрицательное целое число, которое задаёт ограничения для количества элементов списка. Действительный список leaf-list или list **должен** иметь по меньшей мере min-elements элементов

Если оператор min-elements не присутствует, по умолчанию принимается значение 0.

Поведение ограничений зависит от типа ближайшего предка leaf-list или list в дереве схемы, который не является контейнером без присутствия (см. параграф 7.5.1):

- если такого предка нет в дереве схемы, ограничения применяются;
- в противном случае, если предок является узлом case, ограничения применяются при наличии любого другого узла от этого case;
- в остальных случаях ограничения применяются при наличии узла предка.

Далее ограничения применяются в соответствии с правилами раздела 8.

7.7.4. Оператор max-elements

Необязательный оператор max-elements принимает в качестве аргумента положительное целое число или строку unbounded (не ограничено), которые ограничивают число пригодных элементов списка. Действительный список leaf-list или list всегда имеет не более max-elements элементов.

При отсутствии оператора max-elements по умолчанию используется значение unbounded.

Ограничения оператора max-elements применяются в соответствии с правилами раздела 8.

7.7.5. Оператор ordered-by

Оператор ordered-by определяет источник упорядочения элементов списка - пользователь или система. Аргументом оператора может быть строка system или user. При отсутствии оператора по умолчанию применяется system.

Этот оператор игнорируется, если список представляет данные состояния, выходные параметры RPC или содержимое уведомления.

Дополнительная информация приведена в параграфе 7.7.1.

7.7.5.1. Системное упорядочение

Порядок сортировки элементов списка не задан. Реализация может сортировать элементы в удобном для неё порядке. упорядочиваются системой (ordered-by system). Реализациям **следует** применять для одинаковых данных один и тот же порядок, независимо от места расположения данных. Использование определённого порядка позволяет выполнять сравнение с использованием простых средств типа diff.

Этот порядок применяется по умолчанию.

7.7.5.2. Упорядочение пользователем

Элементы списка сортируются в соответствии с порядком, заданным пользователем. Этот порядок устанавливается с помощью специальных атрибутов XML в запросе <edit-config> (см. параграф 7.7.7).

7.7.6. Правила отображения XML

Узел leaf-list представляется последовательностью элементов XML. Локальное имя каждого элемента служит идентификатором leaf-list, а пространством имён служит пространство имён XML модуля (см. параграф 7.1.3).

Значение каждого элемента leaf-list представляется в XML в соответствии с типом и передаётся в элемент XML как символьные данные.

Элементы XML, представляющие элементы leaf-list, **должны** размещаться в порядке, заданном пользователем, если leaf-list имеет атрибут ordered-by user. В противном случае порядок зависит от реализации. Элементы XML, представляющие элементы leaf-list, **могут** чередоваться с элементами братских (sibling) leaf-list, если leaf-list не определяет входные или выходные параметры RPC.

Пример представлен в параграфе 7.7.8.

7.7.7. Операции NETCONF <edit-config>

Элементы leaf-list можно создавать и удалять, но нельзя изменить с помощью <edit-config> путём использования атрибута operation в элементе XML узла leaf-list.

В упорядоченном пользователем leaf-list атрибуты insert и value в пространстве имён YANG XML (параграф 5.3.1) могут служить для управления местом вставки элемента в leaf-list. Они могут применяться в операции create для создания нового элемента в leaf-list, а также в операциях merge или replace для вставки или перемещения элемента.

Атрибут оператора insert может принимать значения first, last, before и after. Для значений before и after **должен** также указываться атрибут value, задающий имеющуюся в leaf-list запись.

Если атрибут insert не задан для операции create, по умолчанию элемент добавляется в конец списка (last).

Если несколько элементов упорядоченного пользователем leaf-list меняются в одном запросе <edit-config>, элементы изменяются по одному в порядке размещения элементов XML в запросе.

В командах <copy-config> и <edit-config> с операцией replace, покрывающей весь список leaf-list, порядок в leaf-list совпадает с порядком элементов XML в запросе.

При обработке сервером NETCONF запроса <edit-config> для узла leaf-list выполняются следующие правила:

- если задана операция merge или replace, узел создаётся при его отсутствии;
- если задана операция create, узел создаётся при его отсутствии, а в случае наличия узла возвращается ошибка data-exists;
- если задана операция delete, узел удаляется при его наличии, а в случае отсутствия узла возвращается ошибка data-missing.

7.7.8. Пример использования

```
leaf-list allow-user  {
    type string;
    description "Список шаблонов имён пользователей для разрешения.";
}
```

Соответствующий экземпляр XML имеет вид

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

Для создания в списке нового элемента с использованием принятой по умолчанию для <edit-config> операции 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="http://example.com/schema/config">
        <services>
          <ssh>
            <allow-user>eric</allow-user>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

Для упорядоченного пользователем списка leaf-list

```
leaf-list cipher  {
    type string;
    ordered-by user;
    description "Список шифров";
}
```

Приведённый ниже фрагмент может служить для вставки нового шифра blowfish-cbc после 3des-cbc

```
<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="http://example.com/schema/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. Оператор list

Оператор list используется для определения внутренних узлов данных в дереве схемы. Узлы списков могут присутствовать в дереве данных во множестве экземпляров. Каждый такой экземпляр называется записью списка. Оператор list принимает один аргумент, являющийся идентификатором, за которым следует блок субоператоров с детальной информацией о списке.

Записи списка однозначно идентифицируются значениями ключей списка, если эти ключи определены.

7.8.1. Субоператоры для list

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
config	7.19.1	0..1
container	7.5	0..n

description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	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	7.7.4	0..1
min-elements	7.7.3	0..1
must	7.5.3	0..n
ordered-by	7.7.5	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
unique	7.8.3	0..n
uses	7.12	0..n
when	7.19.5	0..1

7.8.2. Оператор key

Оператор key, который должен присутствовать в списках, представляющих конфигурацию, и **может** присутствовать в остальных случаях, принимает в качестве аргумента строку, содержащую список из одного или множества разделённых пробелами идентификаторов листьев этого списка. Один идентификатор **недопустимо** включать в несколько ключей. Каждый такой идентификатор листа **должен** указывать на дочерний лист списка. Листья могут определяться непосредственно с субоператорах или в группировках, используемых списком.

Комбинация значений всех листьев, указанных в ключе, служит для однозначной идентификации элемента списка. Все листья ключа **должны** получать значения при создании элемента списка. Таким образом, любые принятые по умолчанию значения в листьях ключа и их типах игнорируются. Операторы mandatory в листьях ключа игнорируются.

Лист, являющийся частью ключа, может быть любого встроенного или производного типа, но **недопустимо** использовать встроенный тип empty.

Все листья ключа в списке **должны** иметь то же значение config, что и сам список.

Синтаксис строки key формально определён правилом key-arg в разделе 12.

7.8.3. Оператор unique

Оператор unique служит задаёт ограничения для пригодных элементов списка и принимает аргумент в форме строки с разделёнными пробелами идентификаторами узлов из списка схемы, которые должны быть заданы в форме потомков (см. правило descendant-schema-nodeid в разделе 12). Каждый из этих идентификаторов узлов схемы **должен** указывать лист.

Если один из указанных листьев представляет данные конфигурации, все указанные листья также **должны** представлять данные конфигурации.

Оператор unique задаёт, что комбинация значений всех экземпляров листьев, указанных в строке аргумента, включая листья с принятыми по умолчанию значениями, **должна** быть уникальной в рамках всех экземпляров элементов списка, в которых указанные листья существуют. Ограничение применяется в соответствии с правилами раздела 8.

Синтаксис строки unique формально определён правилом unique-arg из раздела 12.

7.8.3.1. Пример использования

Для показанного ниже списка

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

приведённая ниже конфигурация будет непригодной

```
<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>
```

Следующая конфигурация приемлема, поскольку элементы списка http и ftp не имеют значений для всех упомянутых листьев и не будут приняты во внимание при выполнении ограничений данных оператором unique.

```
<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. Операторы дочерних узлов списка

Внутри списка операторы container, leaf, list, leaf-list, uses, choice и anyxmls могут служить для определения дочерних узлов списка.

7.8.5. Правила отображения XML

Список представляется набором элементов XML по одному на каждый элемент списка. Локальное имя каждого элемента является идентификатором списка, а пространством имён служит пространство имён XML для модуля (см. параграф 7.1.3).

Узлы ключей списка представляются в форме субэлементов идентификаторов списка в том же порядке, как они определены в операторе key.

Остальная часть дочерних узлов списка представляется в виде субэлементов для элемента списка после ключей. Если список определяет входные или выходные параметры RPC, субэлементы кодируются в том же порядке, как они определены в операторе list. В стальных случаях порядок может быть произвольным.

Все пробельные символы между субэлементами элемента списка не имеют значения и реализация **может** добавлять пробельные символы между субэлементами.

Элементы XML, представляющие элементы списка, должны указываться в заданном пользователем порядке, если список имеет тип ordered-by user. В противном случае порядок зависит от реализации. Элементы XML, представляющие элементы списка, **могут** чередоваться с элементами братских (sibling) списков, если список не определяет входные или выходные параметры RPC.

7.8.6. Операции NETCONF <edit-config>

Элементы списка можно создавать, удалять, заменять и изменять с помощью операции <edit-config> с использованием атрибута operation в элементе XML для списка. В каждом случае значения всех ключей служат для однозначной идентификации элементов списка. Если для элемента списка не заданы все ключи, возвращается ошибка missing-element.

В упорядоченных пользователем списках атрибуты insert и key в пространстве имён YANG XML (параграф 5.3.1) могут служить для управления местом размещения элемента в списке. Они могут применяться в операциях create для вставки новых элементов, а также в операциях merge или replace для вставки для вставки или перемещения элемента.

Атрибут оператора insert может принимать значения first, last, before и after. Для значений before и after должен также указываться атрибут key, задающий имеющийся в списке элемент. Значение атрибута key является ключом, задающим полный идентификатор экземпляра (см. параграф 9.13) для элемента списка.

Если атрибут insert не задан для операции create, по умолчанию элемент добавляется в конец списка (last).

Если несколько элементов упорядоченного пользователем списка меняются в одном запросе <edit-config>, элементы изменяются по одному в порядке размещения элементов XML в запросе.

В командах <copy-config> и <edit-config> с операцией replace, покрывающей весь список, порядок в списке совпадает с порядком элементов XML в запросе.

При обработке сервером NETCONF запроса <edit-config> для узла list выполняются приведённые ниже правила.

- Если задана операция merge или replace, элемент списка создаётся при его отсутствии. Если элемент списка уже существует и имеются атрибуты insert и key, элемент перемещается в соответствии со значениями этих атрибутов. Если элемент списка существует, но атрибутов insert и key нет, элемент списка не перемещается.
- Если задана операция create, элемент списка создаётся при его отсутствии, а в случае наличия элемента возвращается ошибка data-exists.
- Если задана операция delete, элемент списка удаляется при его наличии, а в случае отсутствия элемента возвращается ошибка data-missing.

7.8.7. Пример использования

Для списка

```
list user {
    key "name";
    config true;
    description "Это список пользователей системы.";

    leaf name {
        type string;
```

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

Соответствующий экземпляр XML будет иметь вид

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

Создание нового пользователя 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="http://example.com/schema/config">
        <user nc:operation="create">
          <name>barney</name>
          <type>admin</type>
          <full-name>Barney Rubble</full-name>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

Изменение роли пользователя fred на superuser

```
<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="http://example.com/schema/config">
        <user>
          <name>fred</name>
          <type>superuser</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

Для упорядоченного пользователем списка

```
list user {
    description "Это список пользователей системы.";
    ordered-by user;
    config true;

    key "name";

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

Приведённый ниже фрагмент будет использоваться для добавления нового пользователя barney после пользователя fred

```
<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="http://example.com/schema/config"
           xmlns:ex="http://example.com/schema/config">
```

```
          <user nc:operation="create"
                yang:insert="after"
                yang:key="[ex:name='fred']">
            <name>barney</name>
            <type>admin</type>
            <full-name>Barney Rubble</full-name>
          </user>
        </system>
      </config>
    </edit-config>
  </rpc>
```

Приведённый ниже фрагмент будет служить для размещения пользователя barney перед пользователем fred

```
  <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="http://example.com/schema/config"
            xmlns:ex="http://example.com/schema/config">
          <user nc:operation="merge"
                yang:insert="before"
                yang:key="[ex:name='fred']">
            <name>barney</name>
          </user>
        </system>
      </config>
    </edit-config>
  </rpc>
```

7.9. Оператор choice

Оператор choice определяет набор вариантов, из которых в дереве данных в каждый момент может присутствовать только один. Аргументом оператора является идентификатор, за которым следует блок субоператоров с детальной информацией о выборе. Идентификатор служит для указания узла choice в дереве схемы. В дереве данных узлов choice не существует.

Выбор (choice) включает множество ветвей, каждая из которых определяется субоператором case. Каждая ветвь содержит множество дочерних узлов. Одновременно могут существовать узлы не более, чем из одной ветви.

Дополнительная информация приведена в параграфе 8.3.2.

7.9.1. Субоператоры для choice

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
case	7.9.2	0..n
config	7.19.1	0..1
container	7.5	0..n
default	7.9.3	0..1
description	7.19.3	0..1
if-feature	7.18.2	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	7.19.4	0..1
status	7.19.2	0..1
when	7.19.5	0..1

7.9.2. Оператор case для выбора

Оператор case используется для определения вариантов выбора в операторе choice. Он принимает идентификатор в качестве аргумента, за которым следует блок субоператоров, определяющих варианты.

Идентификатор служит для указания узла case в дереве схемы. Узлов case не существует в дереве данных.

Внутри case могут применяться операторы anyxml, choice, container, leaf, list, leaf-list и uses для определения потомков узла case. Идентификаторы всех дочерних узлов **должны** быть уникальными для всех вариантов case в данном choice. Ниже приведён пример недопустимого выбора.

```
  choice interface-type {     // Это неприемлемо в YANG
      case a {
          leaf ethernet { ... }
      }
      case b {
          container ethernet { ...}
      }
  }
```

Для сокращения оператор case может быть опущен, если вариант содержит один оператор anyxml, choice, container, leaf, list или leaf-list. В таких случаях узел case продолжает существовать в дереве схемы и его идентификатором

служит идентификатор дочернего узла. Идентификаторы узлов схемы (параграф 6.5) всегда должны явно включать идентификаторы узлов case. Выражение

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

эквивалентно

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

Каждый идентификатор case **должен** быть уникальным в рамках choice.

7.9.2.1. Субоператоры для case

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
uses	7.13	0..n
when	7.19.5	0..1

7.9.3. Субоператор default для выбора

Оператор default указывает вариант (case), который следует применять по умолчанию, если нет дочерних узлов ни у одного из вариантов choice. Аргументом служит идентификатор заданного по умолчанию варианта (case). Если оператор default не задан, используемого по умолчанию варианта не будет.

Оператор default **недопустимо** включать в оператор выбора (choice), где для mandatory задано значение true.

Используемый по умолчанию вариант важен лишь при рассмотрении используемых по умолчанию значений в узлах вариантов. Принятые по умолчанию значения для узлов используемого по умолчанию варианта используются в случаях отсутствия каких-либо других вариантов.

Недопустимо наличие обязательных (mandatory) узлов (параграф 3.1), являющихся непосредственными потомками принятого по умолчанию варианта.

Принятые по умолчанию значения дочерних узлов case используются лишь в тех случаях, когда один из таких узлов присутствует в варианте (case) или вариант задан в качестве принятого по умолчанию. Если нет ни одного из дочерних узлов варианта и вариант не задан в качестве принятого по умолчанию, используемые по умолчанию значения дочерних узлов варианта игнорируются.

В приведённом ниже примере выбор (choice) указывает по умолчанию вариант interval и принятое по умолчанию значение будет использоваться, если нет ни одного из листьев daily, time-of-day и manual. Если узел daily присутствует, будет использовано значение, принятое по умолчанию для узла time-of-day.

```
container transfer {
    choice how {
        default interval;
        case interval {
            leaf interval {
                type uint16;
                default 30;
                units minutes;
            }
        }
        case daily {
            leaf daily {
                type empty;
            }
            leaf time-of-day {
                type string;
                units 24-hour-clock;
                default 1am;
            }
        }
        case manual {
            leaf manual {
                type empty;
            }
        }
    }
}
```

7.9.4. Оператор mandatory для выбора

Необязательный оператор mandatory принимает в качестве аргумента строку true или false, задающую ограничения для данных. Если для оператора mandatory установлено значение true, **должен** существовать хотя бы один узел в одном из вариантов choice.

Если оператор не задан, по умолчанию предполагается значение false.

Поведение ограничений зависит от типа ближайшего предка choice в дереве схемы, который не является контейнером без присутствия (см. параграф 7.5.1):

- если предок является узлом case, ограничение применяется при наличии любого другого узла от этого варианта (case);
- в остальных случаях ограничение применяется при наличии узла-предка.

Далее ограничения применяются в соответствии с правилами раздела 8.

7.9.5. Правила отображения XML

Узлы choice и case не видны в XML.

Дочерние узлы выбранного варианта (case) **должны** представляться в том же порядке, как они были определены в операторе case, если это часть определения входных или выходных параметров RPC. В остальных случаях субэлементы могут представляться в любом порядке.

7.9.6. Операции NETCONF <edit-config>

Поскольку в каждый момент может действовать лишь один из вариантов choice, создание узла от одного варианта неявно удаляет все узлы от всех других вариантов. Если операция <edit-config> создаёт узел от case, сервер NETCONF будет удалять все имеющиеся узлы, определённые в других вариантах этого оператора choice.

7.9.7. Пример использования

Для приведённого ниже примера

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

Соответствующий экземпляр XML будет иметь вид

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

Для смены протокола TCP на UDP может служить приведённый ниже фрагмент.

```
<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="http://example.com/schema/config">
        <protocol>
          <udp nc:operation="create"/>
        </protocol>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.10. Оператор anyxml

Оператор anyxml определяет внутренний узел в дереве схемы. Оператор принимает идентификатор узла в качестве аргумента, а за ним следует блок субоператоров с информацией anyxml.

Оператор anyxml служит для представления неизвестного блока данных XML, на который не накладывается каких-либо ограничений. Это может быть полезно, например, в откликах RPC. Примером может служить параметр <filter> в операции <get-config> протокола NETCONF.

Узел anyxml не может быть дополнен (см. параграф 7.15).

Поскольку применение anyxml ограничивает манипуляции с содержимым, **рекомендуется** не использовать оператор anyxml для определения данных конфигурации.

Узел anyxml отсутствует в дереве данных или содержится там в одном экземпляре.

7.10.1. Субоператоры для anyxml

Субоператор	Параграф	Число элементов
config	7.19.1	0..1
description	7.19.3	0..1
if-feature	7.18.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
when	7.19.5	0..1

7.10.2. Правила отображения XML

Узел anyxml представляется в виде элемента XML. Локальное имя элемента является идентификатором anyxml, а пространством имён служит пространство имён XML для модуля (см. параграф 7.1.3). Значение узла anyxml представляется в качестве содержимого XML для этого элемента.

Отметим, что любые префиксы XML, используемые в представлении, являются локальными для представления каждого экземпляра. Это означает возможность различного представления XML в разных реализациях.

7.10.3. Операции NETCONF <edit-config>

Узел anyxml рассматривается, как «непрозрачный» (opaque) блок данных. Эти данные могут изменяться лишь целиком.

Любые атрибуты operation, представленные в субэлементах узла anyxml, игнорируются сервером NETCONF.

При обработке сервером NETCONF запроса <edit-config> для узла anyxml выполняются приведённые ниже правила.

- Если задана операция merge или replace, узел создаётся при его отсутствии, а в качестве значения устанавливается содержимое XML узла anyxml в данных XML для RPC.
- Если задана операция create, узел создаётся при его отсутствии, а в качестве значения устанавливается содержимое XML узла anyxml в данных XML для RPC. В случае наличия узла возвращается ошибка data-exists.
- Если задана операция delete, узел удаляется при его наличии, а в случае отсутствия узла возвращается ошибка data-missing.

7.10.4. Пример использования

Для оператора anyxml

```
anyxml data;
```

ниже приведены два варианта корректного представления одного значения anyxml.

```
<data xmlns:if="http://example.com/ns/interface">
  <if:interface>
    <if:ifIndex>1</if:ifIndex>
  </if:interface>
</data>

<data>
  <interface xmlns="http://example.com/ns/interface">
    <ifIndex>1</ifIndex>
  </interface>
</data>
```

7.11. Оператор grouping

Оператор grouping служит для определения блока узлов, который может многократно применяться локально в модуле или субмодуле, а также в импортирующих данный модуль других модулях (в соответствии с правилами параграфа 5.5). Оператор принимает в качестве аргумента идентификатор, за которым следует блок субоператоров с информацией о группировке.

Оператор grouping не задаёт определения данных и по этой причине не определяет каких-либо узлов в дереве схемы.

Группировка напоминает структуры (structure) или записи (record) в традиционных языках программирования.

После определения группировки на неё можно ссылаться в операторе uses (см. параграф 7.12). Группировке **недопустимо** ссылаться на самое себя ни напрямую, ни опосредовано через цепочку других группировок.

Если группировка определена на верхнем уровне модуля или субмодуля YANG, идентификатор группировки **должен** быть уникальным внутри модуля.

Группировка - это не просто механизм текстовой подстановки, она также определяет набор узлов. Идентификаторы узлов группировки преобразуются применительно к области, в которой группировка была определена, а не той, где она применяется. Отображения префиксов, имена типов и группировок, использование расширений оцениваются в иерархии, где был применён оператор grouping. Для расширений это означает, что они применяются к самой группировке, а не к использующему узлу.

7.11.1. Субоператоры для grouping

Субоператор	Параграф	Число элементов
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1

grouping	7.11	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
uses	7.12	0..n

7.11.2. Пример использования

```
import ietf-inet-types {
    prefix "inet";
}

grouping endpoint {
    description "Повторно используемая группа конечных точек.";
    leaf ip {
        type inet:ip-address;
    }
    leaf port {
        type inet:port-number;
    }
}
```

7.12. Оператор uses

Оператор uses применяется для ссылок на определения группировок (grouping) и принимает один аргумент, задающий имя группировки.

Эффект ссылки uses на группировку заключается в том, что определённые группировкой узлы копируются в текущее дерево схемы, а затем обновляются в соответствии с операторами refine и augment.

Идентификаторы, определённые в группировке, не привязаны к пространству имён, пока содержимое группировки не будет добавлено в дерево схемы с помощью оператора uses, который не присутствует внутри данной группировки, после чего оно привязывается к пространству имён текущего модуля.

7.12.1. Субоператоры для uses

Субоператор	*Параграф*	*Число элементов*
augment	7.15	0..n
description	7.19.3	0..1
if-feature	7.18.2	0..n
reference	7.19.4	0..1
refine	7.12.2	0..n
status	7.19.2	0..1
when	7.19.5	0..1

7.12.2. Оператор refine

Некоторые из свойств каждого узла в группировке могут уточняться с помощью оператора refine, аргументом которого является строка, указывающая узел в этой группировке. Такой узел называется целью уточнения. Если узел в группировке не указан в качестве цели уточнения в операторе refine, он сохраняется без изменений и остаётся в точности таким, каким был определён в группировке.

Строка аргумента представляет собой идентификатор узла-потомка в схеме (см. параграф 6.5).

Оператор позволяет выполнить перечисленные ниже уточнения.

- Узел leaf или choice может получить используемое по умолчанию значение или обновить имеющееся.
- Любой узел может получить специализированную строку описания (description).
- Любой узел может получить специализированную строку ссылки (reference).
- Любой узел может получить другой оператор config.
- Узел leaf, anyxml или choice может получить другой оператор mandatory.
- Контейнер может получить оператор presence.
- Узел leaf, leaf-list, list, container или anyxml может получить дополнительные выражения must.
- Узел leaf-list или list может получить другие операторы min-elements или max-elements.

7.12.3. Правила отображения XML

Каждый узел в группировке представляется, как будто он был определён здесь (inline), даже если он импортирован из другого модуля с иным пространством имён XML.

7.12.4. Пример использования

Для использования группировки endpoint, определённой в параграфе 7.11.2, при определении сервера HTTP в каком-то другом модуле можно задать

```
import acme-system {
    prefix "acme";
}
```

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

Соответствующий экземпляр XML будет иметь вид

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

Если на сервере HTTP по умолчанию следует применять порт 80, можно добавить оператор default.

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

Если нужно определить список серверов, каждый из которых имеет ip и port в качестве ключей, можно задать

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

Ниже приведён пример с ошибкой.

```
container http-server {
    uses acme:endpoint;
    leaf ip {          // недопустимо - идентификатор ip применяется дважды
        type string;
    }
}
```

7.13. Оператор rpc

Оператор rpc служит для определения операций NETCONF RPC. Он принимает один аргумент в форме идентификатора, за которым следует блок субоператоров, определяющих операцию вызова удалённой процедуры. Аргумент является именем RPC и используется в качестве имени элемента для <rpc>, как указано группой подстановки rpcOperation в [RFC4741].

Оператор rpc определяет узел rpc в дереве схемы. Для узла rpc в схеме также определяются дочерние узлы input и output. Эти узлы определяются в пространстве имён модуля.

7.13.1. Субоператоры для rpc

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	0..n
input	7.13.2	0..1
output	7.13.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n

7.13.2. Оператор input

Необязательный оператор input служит для определения входных параметров операции RPC и не имеет аргументов. Субоператоры для input определяют узлы-потомки для узла input операции RPC.

Если лист в дереве input имеет оператор mandatory со значением true, этот лист **должен** присутствовать в вызове NETCONF RPC. В противном случае сервер **должен** возвращать ошибку NETCONF.

Если лист в дереве input имеет принятое по умолчанию значение, сервер NETCONF **должен** использовать это значение в случаях, описанных в параграфе 7.6.1. В таких ситуациях сервер **должен** вести себя так, будто при вызове NETCONF RPC принятое по умолчанию значение листа было передано в качестве входного.

Если оператор config присутствует для любого узла в дереве input, оператор config игнорируется.

Если узел имеет оператор when, который даёт значение false, такой узел **недопустимо** включать в дерево input.

7.13.2.1. Субоператоры для input

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n

grouping	7.11	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
typedef	7.3	0..n
uses	7.12	0..n

7.13.3. Оператор output

Необязательный оператор output служит для определения выходных параметров операции RPC и не имеет аргументов. Субоператоры для output определяют узлы-потомки узла output определяемой операции.

Если лист дерева имеет оператор mandatory со значением true, лист **должен** присутствовать в отклике NETCONF RPC.

Если лист в дереве output имеет принятое по умолчанию значение, клиент NETCONF **должен** использовать это значение в случаях, описанных в параграфе 7.6.1. В таких ситуациях клиент **должен** вести себя так, будто принятое по умолчанию значение листа присутствует в отклике NETCONF RPC.

Если оператор config присутствует для любого узла в дереве output, оператор config игнорируется.

Если любой из узлов имеет оператор when, выражение которого даёт значение false, такой узел **недопустимо** включать в дерево output.

7.13.3.1. Субоператоры для output

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.11	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
typedef	7.3	0..n
uses	7.12	0..n

7.13.4. Правила отображения XML

Узел rpc представляется как дочерний элемент XML элемента <rpc> в соответствии с группой подстановки rpcOperation [RFC4741]. Локальное имя элемента является идентификатором rpc, а его пространство имён - пространством имён XML для модуля (см. параграф 7.1.3).

Входные параметры представляются дочерними элементами XML для элемента XML узла rpc в том же порядке, как они были определены в операторе input.

Если после успешного вызова RPC не было возвращено выходных параметров, <rpc-reply> содержит только элемент <ok/>, определённый в [RFC4741]. Если выходные параметры присутствуют, они кодируются как дочерние элементы элемента <rpc-reply>, определённого в [RFC4741], с тем же порядком, который был определён в операторе output.

7.13.5. Пример использования

Приведённый ниже пример определяет операцию RPC.

```
module rock {
    namespace "http://example.net/rock";
    prefix "rock";

    rpc rock-the-house {
        input {
            leaf zip-code {
                type string;
            }
        }
    }
}
```

Соответствующие экземпляры XML для полных rpc и rpc-reply приведены ниже.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="http://example.net/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>
```

7.14. Оператор notification

Оператор notification служит для определения уведомлений NETCONF. Оператор принимает в качестве аргумента идентификатор уведомления, за которым следует блок субоператоров с деталями уведомления. Оператор notification определяет уведомление в дереве схемы.

Если лист в дереве уведомления имеет оператор mandatory со значением true, этот лист **должен** присутствовать в экземплярах уведомления NETCONF.

Если лист в дереве уведомления имеет принятое по умолчанию значение, клиент NETCONF **должен** использовать это значение в случаях, описанных в параграфе 7.6.1. К таких ситуациях клиент **должен** вести себя так, будто лист присутствует в экземпляре уведомления NETCONF с принятым по умолчанию значением.

Поскольку дерево notification не включается в какое-либо хранилище данных, все операторы config для узлов дерева notification игнорируются.

Если оператор config присутствует для любого узла в дереве notification, оператор config игнорируется.

7.14.1. Субоператоры для notification

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
grouping	7.11	0..n
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
typedef	7.3	0..n
uses	7.12	0..n

7.14.2. Правила отображения XML

Узел notification, определённый на верхнем уровне модуля, кодируется как дочерний элемент XML для элемента <notification>, определённого в NETCONF Event Notifications [RFC5277]. Локальное имя элемента является идентификатором уведомления, а его пространством имён является пространство имён XML для модуля (см. параграф 7.1.3).

7.14.3. Пример использования

Приведённый ниже пример определяет уведомление.

```
module event {
    namespace "http://example.com/event";
    prefix "ev";
    notification event {
        leaf event-class {
            type string;
        }
        anyxml reporting-entity;
        leaf severity {
            type string;
        }
    }
}
```

Соответствующий экземпляр элемента XML для полного уведомления имеет вид

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-07-08T00:01:00Z</eventTime>
  <event xmlns="http://example.com/event">
    <event-class>fault</event-class>
    <reporting-entity>
      <card>Ethernet0</card>
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

7.15. Оператор augment

Оператор augment позволяет модулю или субмодулю добавить узлы к дереву схемы, определённому во внешнем модуле, текущем модуле и его субмодулях, или к узлам из группировки в операторе uses. Аргументом оператора служит строка, указывающая узел в дереве схемы. Этот узел называется целью добавления (augment's target). Целевой узел **должен** быть узлом типа container, list, choice, case, input, output и notification. К нему добавляются узлы, указанные с субоператорах, следующих за оператором augment.

Строка дополнения является идентификатором узла схемы (см. параграф 6.5). Если оператор augment относится к верхнему уровню модуля или субмодуля, **должна** применяться абсолютная форма (определена правилом absolute-schema-nodeid раздела 12) идентификатора узла схемы. Если augment является субоператором для uses, **должна** применяться наследуемая (descendant) форма (определена правилом descendant-schema-nodeid в разделе 12).

Если целевой узел имеет тип container, list, case, input, output или notification, внутри оператора augment могут применяться субоператоры container, leaf, list, leaf-list, uses и choice.

Если целевой узел является контейнером или списком, внутри оператора augment могут применяться субоператоры action и notification.

Если целевой узел является выбором (choice), внутри оператора augment могут применяться субоператоры case в полной или сокращённой (см. параграф 7.9.2) форме.

Если целевой узел является другим модулем, добавляемым узлам **недопустимо** быть обязательными (параграф 3.1).

Оператору augment **недопустимо** добавлять в целевой модуль множество узлов с одним именем из одного модуля.

7.15.1. Субоператоры для augment

Субоператор	*Параграф*	*Число элементов*
anyxml	7.10	0..n
case	7.9.2	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.19.3	0..1
if-feature	7.18.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.19.4	0..1
status	7.19.2	0..1
uses	7.12	0..n
when	7.19.5	0..1

7.15.2. Правила отображения XML

Все узлы данных, определённые в операторе augment, задаются как элементы XML, определённые в пространстве имён XML модуля, где указан оператор augment.

При добавлении узла дополняющие потомки кодируются в виде субэлементов дополняемого узла в любом порядке.

7.15.3. Пример использования

В пространстве имён http://example.com/schema/interfaces

```
container interfaces {
    list ifEntry {
        key "ifIndex";

        leaf ifIndex {
            type uint32;
        }
        leaf ifDescr {
            type string;
        }
        leaf ifType {
            type iana:IfType;
        }
        leaf ifMtu {
            type int32;
        }
    }
}
```

В пространстве имён http://example.com/schema/ds0

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

Соответствующий экземпляр XML будет иметь вид

```
<interfaces xmlns="http://example.com/schema/interfaces"
            xmlns:ds0="http://example.com/schema/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>
```

В другом примере предполагается наличие выбора (choice), определённого в параграфе 7.9.7. Приведённая ниже конструкция может использоваться для расширения протокольного определения

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

```
        }
    }
```

Соответствующий экземпляр XML будет иметь вид

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

или

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

7.16. Оператор identity

Оператор identity служит для определения нового уникального в глобальном масштабе абстрактного отождествления без конкретного типа. Единственной целью создания такого отождествления является обозначение его имени, семантики и существования. Отождествление может быть определено «с нуля» или произведено на основе одного или нескольких базовых отождествлений. Аргументом identity является идентификатор, указывающий имя этого отождествления, за которым следует блок субоператоров с информацией об отождествлении.

Для определения отождествлений в модели данных может применяться встроенный типа данных identityref (см. параграф 9.10).

7.16.1. Субоператоры для identity

Субоператор	*Параграф*	*Число элементов*
base	7.16.2	0..n
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1

7.16.2. Оператор base

Необязательный оператор base принимает в качестве аргумента строку имени существующего отождествления, на основе которого создаётся производное. Если оператор base не указан, новое отождествление создаётся «с нуля».

Если в базовом имени присутствует префикс, это говорит о том, что базовое отождествление определено в модуле, который импортирован с этим префиксом, или в локальном модуле, если тот использует указанный префикс. В остальных случаях отождествление с указанным именем **должно** быть определено в текущем модуле или его субмодулях.

Поскольку субмодули не могут включать родительский модуль, любые отождествления в модуле, которые нужно открыть для субмодулей, **должны** определяться в субмодуле. Другие субмодули могут включать этот субмодуль и видеть определение identity.

Отождествлению **недопустимо** ссылаться на себя напрямую или косвенно через цепочку других отождествлений.

7.16.3. Пример использования

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

        identity crypto-alg {
            description
               "Базовое отождествление, из которого выводятся все
                криптоалгоритмы.";
        }
    }

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

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

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

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

7.17. Оператор extension

Оператор extension позволяет определять новые операторы для языка YANG. Определение нового оператора может импортироваться и использоваться другими модулями.

Аргументом оператора extension является идентификатор, который будет ключевым словом расширения, а за ним должен следовать блок субоператоров с информацией о расширении. Целью оператора extension является определение нового ключевого слова, которое может импортироваться и применяться другими модулями.

Расширение может применяться подобно обычным операторам YANG с именем оператора, за которым следует аргумент, если он был определён оператором extension, а также может следовать блок субоператоров. Имя оператора создаётся путём объединения префикса модуля, в котором было определено расширение, двоеточия (:) и ключевого слова расширения без пробелов между ними. Субоператоры расширения определяются оператором extension, с использованием механизмов, выходящих за рамки данной спецификации. Синтаксически субоператоры **должны** быть операторами YANG, включая расширения, определённые с использованием операторов extension. Операторы YANG в расширениях должны следовать синтаксическим правила раздела 12.

7.17.1. Субоператоры для extension

Субоператор	*Параграф*	*Число элементов*
argument	7.17.2	0..1
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1

7.17.2. Оператор argument

Необязательный оператор argument принимает в качестве аргумента строку имени аргумента для ключевого слова (расширения). Если оператор argument не задан, ключевое слово будет использоваться без аргументов.

Имя аргумента используется в отображении YIN, где оно служит атрибутом XML или именем элемента в зависимости от субоператора yin-element в этом операторе argument.

7.17.2.1. Субоператор для argument

Субоператор	*Параграф*	*Число элементов*
yin-element	7.17.2.2	0..1

7.19.2.2. Оператор yin-element

Необязательный оператор yin-element принимает в качестве аргумента строку true или false. Этот оператор показывает отображается ли argument в элемент или атрибут XML при преобразовании в YIN (см. раздел 11).

Если оператор yin-element не задан, по умолчанию предполагается значение false.

7.17.3. Пример использования

Определение расширения может иметь вид

```
module my-extensions {
  ...

  extension c-define {
    description
      "Принимает в качестве аргумента строку имени.
       Обеспечивает использование генератором кода этого имени в
       #define.";
    argument "name";
  }
}
```

Использование расширения может иметь вид

```
module my-interfaces {
  ...
  import my-extensions {
    prefix "myext";
  }
  ...

  container interfaces {
    ...
    myext:c-define "MY_INTERFACES";
  }
}
```

7.18. Операторы, связанные с соответствием спецификации

В этом разделе определены операторы, связанные с совместимостью, как описано в параграфе 5.6.

7.18.1. Оператор feature

Оператор feature служит для определения механизма, с помощью которого часть схемы помечается, как условная. Определяется имя функции, на которое можно потом ссылаться в операторах if-feature (см. параграф 7.18.2). Узлы схемы, помеченные оператором if-feature, игнорируются сервером, если он не поддерживает заданное в операторе feature выражение. Это позволяет сделать часть модуля YANG условной, в зависимости от возможностей сервера. Модель может представлять возможности сервера внутри себя, что делает её более эффективной и позволяет устройствам выступать в различных ролях.

Аргументом оператора feature является имя новой функции (возможности), которое выбирается в соответствии с правилами для идентификаторов, приведёнными в параграфе 6.2. Это имя используется в операторах if-feature для привязки узлов схемы к данной функции.

В приведённом ниже примере функция local-storage представляет способность устройства локально сохранять сообщения syslog в том или ином его хранилище. Эта функция используется для того, чтобы сделать лист local-storage-limit условным, в зависимости от наличия локального хранилища. Если устройство не укажет поддержку этой функции, узел local-storage-limit не будет поддерживаться.

```
module syslog {
    ...
    feature local-storage {
        description
            "Это свойство означает поддержку устройством локального
             хранилища (память, флэш или диск), которое может служить
             для записи сообщений syslog.";
    }

    container syslog {
        leaf local-storage-limit {
            if-feature local-storage;
            type uint64;
            units "kilobyte";
            config false;
            description
                "Объем локального хранилища, который может быть
                 использован для сообщений syslog.";
        }
    }
}
```

Оператор if-feature может использоваться в разных местах в синтаксисе YANG. Определения с пометкой if-feature игнорируются, если устройство не поддерживает данную функцию.

Функции (feature) **недопустимо** ссылаться на себя ни напрямую, ни опосредованно через цепочку других функций.

Для того, чтобы устройство поддерживало функцию, зависящую от каких-либо других функций (т. е. функцию, включающую один или несколько субоператоров if-feature), это устройство **должно** поддерживать все связанные зависимостями функции.

7.18.1.1. Субоператоры для feature

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
if-feature	7.18.2	0..n
reference	7.19.4	0..1
status	7.19.2	0..1

7.18.2. Оператор if-feature

Оператор if-feature делает родительский оператор условным. Аргументом является имя функции, заданное оператором feature. Родительский оператор выполняется сервером в том случае, когда он поддерживает эту возможность. При наличии префикса в имени функции, он указывает возможность, определённую в модуле, импортированном с этим префиксом, или локальном модуле, если префикс соответствует префиксу локального модуля. В остальных случаях соответствующее имя **должно** быть определено в текущем модуле или включённом субмодуле.

Поскольку субмодули не могут включать родительский модуль, все функции в модуле, которые нужно открыть для субмодулей, **должны** быть определены в субмодуле. Субмодули тогда могут включать этот субмодуль для получения доступа к определению функции.

7.18.3. Оператор deviation

Оператор deviation определяет иерархию модуля, которую устройство полностью не реализует. Аргументом оператора служит строка с идентификатором узла в дереве схемы, который отклоняется от заданного модулем поведения. Этот узел называется целевым узлом отклонения. Содержимое оператора deviation указывает детали отклонения.

Строка аргумента указывает абсолютный идентификатор узла схемы (см. параграф 6.5).

Отклонения показывают в чем устройство или класс устройств отклоняется от стандарта. Это означает, что отклонения никогда **не должны** быть частью опубликованного стандарта, поскольку они обеспечивают механизм информирования об отклонении реализации от стандарта.

Устройствам настоятельно рекомендуется избегать отклонений и они **должны** использоваться лишь в качестве последнего шанса. Уведомление клиентов об отклонении от стандарта не заменяет собой следование стандартам. Устройство, поведение которого отклоняется от модуля, не будет полностью совместим с этим модулем.

Однако в некоторых случаях конкретное устройство может просто не иметь аппаратных или программных компонент для поддержки отдельных частей стандартного модуля. В таких случаях устройство может попытаться выполнить не поддерживаемую функцию и возвратить клиенту сообщение об ошибке или просто игнорировать запросы. Оба варианта не будут приемлемыми.

Взамен этих вариантов YANG позволяет серверам указать не поддерживаемые части базового модуля или показать использование для их поддержки другого синтаксиса. Делается это с помощью оператора deviation.

7.18.3.1. Субоператоры для deviation

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
deviate	7.18.3.2	1 - n
reference	7.19.4	0..1

7.18.3.2. Оператор deviate

Оператор deviate указывает отклонение реализации устройства для целевого узла от исходного определения. Аргумент оператора может принимать значение not-supported, add, replace или delete.

Значение not-supported показывает, что целевой узел не реализован на этом устройстве.

Значение add добавляет свойства целевому узлу. Свойства для добавления указываются субоператорами оператора deviate. Если свойство может появляться только один раз, его существование в целевом узле **недопустимо**.

Значение replace заменяет свойства целевого узла. Свойства для замены указываются субоператорами оператора deviate. Заменяемое свойство **должно** существовать у целевого узла.

Значение delete удаляет свойства целевого узла. Свойства для удаления указываются субоператорами оператора deviate. Ключевое слово субоператора **должно** соответствовать ключевому слову в целевом узле, а строка аргумента **должна** совпадать с соответствующей строкой аргумента ключевого слова в целевом узле.

Субоператор	*Параграф*	*Число элементов*
config	7.19.1	0..1
default	7.6.4	0..n
mandatory	7.6.5	0..1
max-elements	7.7.4	0..1
min-elements	7.7.3	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

7.20.3.3. Пример использования

В этом примере устройство информирует клиентское приложение о том, что он не поддерживает службу daytime в стиле RFC 867.

```
deviation /base:system/base:daytime {
    deviate not-supported;
}
```

В приведённом ниже примере устанавливается используемое сервером по умолчанию значение для листа, у которого такого значения не определено.

```
deviation /base:system/base:user/base:type {
    deviate add {
        default "admin"; // новые пользователи по умолчанию admin
    }
}
```

В следующем примере сервер ограничивает число серверов имён до 3.

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

Если исходное определение имело вид

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

устройство может удалить ограничение must

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

7.19. Субоператоры общего назначения

В этом разделе описаны субоператоры, используемые несколькими другими операторами.

7.19.1. Оператор config

Оператор config принимает в качестве аргумента строку true или false. Если config имеет аргумент true, определение представляет конфигурацию. Узлы данных, представляющие конфигурацию, являются частью хранилища конфигурации. Узлы данных, представляющие конфигурацию, будут частью отклика на запрос <get-config> и могут передаваться в запросах <copy-config> или <edit-config>.

Если config имеет аргумент false, определение представляет данные состояния. Узлы данных, представляющих состояние, будут частью отклика на запрос <get>, но не на запрос <get-config> и не могут включаться в запросы <copy-config> или <edit-config>.

Если оператор config не задан, по умолчанию используется значение config родительского узла схемы. Если родительским узлом является case, значение совпадает со значением родителя варианта case.

Если верхний узел на включает config, по умолчанию используется значение true.

Если для узла config имеет значение false, ни один из нижележащих узлов не может иметь config со значением true.

7.19.2. Оператор status

Оператор status принимает в качестве аргумента одно из значений current, deprecated или obsolete:

- current означает, что определений является текущим и действует;
- deprecated указывает устаревшее определение, которое, тем не менее, разрешено использовать для обеспечения совместимости с имеющимися старыми реализациями;
- obsolete указывает отменённое определение, которое **не следует** применять в новых реализациях и/или может быть удалено.

Если оператор status не задан, по умолчанию предполагается значение current.

Определению типа current **недопустимо** ссылаться на определения типа deprecated или obsolete в том же модуле.

Определению типа deprecated **недопустимо** ссылаться на определения типа obsolete в том же модуле.

Например, приведённое ниже определение недействительно.

```
typedef my-type {
  status deprecated;
  type int32;
}

leaf my-leaf {
  status current;
  type my-type; // не пригоден, поскольку my-type отменен
}
```

7.19.3. Оператор description

Оператор description принимает в качестве аргумента строку, содержащую предназначенное для человека текстовое описание данного определения. Текст приводится на языке (языках) выбранном разработчиком модуля. **Рекомендуется** использовать язык, широко распространённый среди сетевых администраторов, которые будут использовать модуль.

7.19.4. Оператор reference

Оператор reference принимает в качестве аргумента строку, содержащую предназначенную для человека ссылку на внешний документ - другой модуль, который определяет связанную с данным информацию для управления, или документ с дополнительной информацией, относящейся к данному определению.

Например, typedef для типа данных uri может иметь вид

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

7.19.5. Оператор when

Оператор when делает условным родительский оператор определения данных. Узел, определённый родительским оператором, будет действительным только при выполнении условия, заданного оператором when. Аргументом оператора является выражением XPath (см. параграф 6.4), которое служит для формального задания условий. Если выражение XPath даёт значение true для конкретного экземпляра, узел, определённый родительским оператором, будет действительным.

Дополнительная информация приведена в параграфе 8.3.2.

Выражение XPath концептуально оценивается в описанном ниже контексте в дополнение к определению параграфа 6.4.1

- Если оператор when является потомком оператора augment, узлом контекста является целевой узел дополнения (augment) в дереве данных, когда целевой узел является узлом данных. В противном случае узлом контекста является узел ближайшего предка целевого узла, который является узлом данных.
- Если оператор when является потомком оператора uses, choice или case, узлом контекста является ближайший предок узла uses, choice или case n, который является узлом данных.
- Если оператор when является потомком любого оператора определения, узлом контекста является узел определения данных в дереве данных.
- Доступное дерево состоит из всех узлов дерева данных и всех листьев, в которых используются принятые по умолчанию значения (параграф 7.6.1).

Доступное дерево зависит от узла контекста, как указано ниже.

- Если узел контекста представляет конфигурацию, дерево является данными из хранилища NETCONF, где существует узел контекста. Корневой узел XPath имеет в качестве потомков все узлы конфигурационных данных верхнего уровня во всех модулях.
- Если узел контекста представляет данные состояния, дерево является всеми данными на устройстве и в хранилище <running/>. Корневой узел XPath имеет в качестве потомков все узлы данных верхнего уровня во всех модулях.

- Если узел контекста представляет содержимое уведомления, дерево является документом экземпляра XML для уведомления. Корневой узел XPath имеет в качестве единственного потомка элемент, представляющий уведомление.
- Если узел контекста представляет входные параметры RPC, дерево является документом экземпляра RPC XML. Корневой узел XPath имеет в качестве единственного потомка элемент, представляющий операцию RPC.
- Если узел контекста представляет выходные параметры RPC, дерево является документом экземпляра отклика RPC. Корневой узел XPath имеет в качестве потомков элементы, представляющие выходные параметры RPC.

Результат выражения XPath преобразуется в логическое значение с использованием стандартных правил XPath.

Отметим, что выражение XPath проверяется концептуально. Это означает, что реализация не обязана использовать оценщик XPath на сервере. Оператор when может быть реализован в отдельном коде.

8. Ограничения

8.1. Ограничения для данных

Некоторые операторы YANG задают ограничения для данных. Эти ограничения реализуются разными способами в зависимости от от типа данных, определяемого оператором.

- Ограничение, заданное для данных конфигурации, **должно** выполняться в дереве данных конфигурации.
- Ограничение для данных состояния **должно** выполняться в откликах на операции <get> без фильтра.
- Ограничение для содержимого уведомления **должно** выполняться в любом экземпляре уведомления.
- Если ограничение задано для входных параметров RPC, оно **должно** выполняться при вызове RPC.
- Если ограничение задано для выходных параметров RPC, оно **должно** выполняться в откликах RPC.

8.2. Иерархия ограничений

Условия на родительских узлах влияют на ограничения дочерних узлов как естественное следствие иерархии узлов. Ограничения must, mandatory, min-elements и max-elements не применяются, если родительский узел имеет условие when или if-feature, которое не выполняется на устройстве.

В приведённом примере ограничение mandatory для листа longitude не будет применяться на устройствах без функции has-gps.

```
container location {
    if-feature has-gps;
    leaf longitude {
        mandatory true;
        ...
    }
}
```

8.3. Модель применения ограничений

Для данных конфигурации имеется три «окна», где ограничения **должны** применяться:

- в процессе анализа содержимого (payload) RPC;
- в процессе выполнения операций NETCONF;
- в процессе проверки пригодности (validation).

Каждый из этих вариантов рассматривается в последующих параграфах.

8.3.1. Анализ данных

Полученное содержимое RPC должно иметь корректный формат XML, а также следовать иерархии и правилам, определяемым моделями, которые устройство реализует.

- Если значение данных листа (leaf) не соответствует ограничениям для листа, включая определённые свойствами range, length и pattern, сервер **должен** ответить сообщением <rpc-error> с тегом <error-tag> invalid-value, а также error-app-tag (параграф 7.5.4.2) и error-message (параграф 7.5.4.1), связанными с ограничениями, если они имеются.
- Если все ключи элемента списка отсутствуют, сервер **должен** передать <error-tag> с missing-element в сообщении <rpc-error>.
- Если присутствуют данные для нескольких вариантов (case) оператора выбора choice, сервер **должен** передать <error-tag> с bad-element в сообщении <rpc-error>.
- Если данные для узла имеют метку if-feature, а выражение if-feature даёт значение false на устройстве, сервер **должен** передать <error-tag> с unknown-element в сообщении <rpc-error>.
- Если присутствуют данные для узла с оператором when, который даёт значение false, сервер **должен** передать <error-tag> с unknown-element в сообщении <rpc-error>.
- Если при обработке вставки узла значения атрибутов before и after не подходят для типа ключевых листьев, сервер **должен** передать <error-tag> с bad-attribute в сообщении <rpc-error>.
- Если атрибуты before и after появляются в каком-либо списке, для которого свойство ordered-by имеет значение user, сервер **должен** передать <error-tag> с unknown- attribute в сообщении <rpc-error>.

8.3.2. Обработка NETCONF <edit-config>

После анализа входных данных сервер NETCONF выполняет операцию <edit-config>, применяя данные к хранилищу конфигурации. В течение этой обработки **должны** детектироваться следующие ошибки:

- запросы на удаление не существующих данных;
- запросы на создание уже имеющихся данных;
- запросы вставки с параметрами before или after, указывающими отсутствующие элементы.

В процессе обработки <edit-config> выполняются описанные ниже условия.

- Если операция NETCONF создаёт узлы данных от оператора выбора choice, все существующие узлы из других вариантов case удаляются сервером.
- Если операция NETCONF меняет узлы данных так, что выражение when на каком-либо узле принимает значения false, такой узел удаляется сервером.

8.3.3. Проверка пригодности

По завершении обработки хранилища данных окончательное содержимое **должно** соответствовать всем ограничениям пригодности. Эта проверка выполняется в разное время в зависимости от хранилища данных. Если хранилище является рабочим (<running/>) или стартовым (<startup/>), ограничения должны применяться в конце операции <edit-config> или <copy-config>. Если хранилище является «кандидатом» (<candidate/>), применение ограничений откладывается до вызова операции <commit> или <validate>.

- Все ограничения must **должны** давать значение true.
- Все ограничения ссылочной целостности, определённые операторами path **должны** выполняться.
- Все ограничения unique для списков **должны** выполняться.
- Ограничения min-elements и max-elements применяются для узлов list и leaf-list.

9. Встроенные типы

Язык YANG имеет набор встроенных типов, похожий на используемые в языках программирования, но с некоторыми отличиями, обусловленными специальными требованиями управления информационными моделями.

Могут определяться дополнительные производные типы на основе встроенных и других производных типов. Производные типы могут использовать субтипы для формального ограничения возможных значений.

Различные встроенные типы и производные от них позволяют организовать разные субтипы для ограничения размера и соответствия строкам регулярных выражений (параграфы 9.4.4 и 9.4.6), а также ограничения диапазонов числовых значений (параграф 9.2.4).

Лексическое представление значений некоторых типов применяется в сообщениях NETCONF и при задании принятых по умолчанию значений и числовых диапазонов в модулях YANG.

9.1. Каноническое представление

Для большинства типов существует одно каноническое представление значений данного типа. Некоторые типы разрешают множество лексических представлений одного и того же значения, например, положительное число 17 можно записать как +17 или 17. Реализации **должны** поддерживать все лексические представления, заданные в этом документе.

Когда сервер передаёт NETCONF данные, он **должен** использовать каноническую форму.

Некоторые типы имеют лексическое представление, которое зависит от контекста XML, в котором они появляются. Такие типы не имеют канонической формы.

9.2. Целочисленные встроенные типы

Язык YANG использует встроенные целочисленные типы int8, int16, int32, int64, uint8, uint16, uint32 и uint64. Они представляют числа разной размерности со знаком или без него:

int8 представляет целые числа от -128 до 127, включительно;

int16 представляет целые числа от -32768 до 32767, включительно;

int32 представляет целые числа от -2147483648 до 2147483647, включительно;

int64 представляет целые числа от -9223372036854775808 до 9223372036854775807, включительно;

uint8 представляет целые числа от 0 до 255, включительно;

uint16 представляет целые числа от 0 до 65535, включительно;

uint32 представляет целые числа от 0 до 4294967295, включительно;

uint64 представляет целые числа от 0 до 18446744073709551615, включительно.

9.2.1. Лексическое представление

Целое число лексически представляется необязательным знаком (+ или -), за которым следуют десятичные цифры. Если знак не указан, предполагается +.

Для удобства при задании используемого по умолчанию целочисленного значения в модуле YANG может применяться другое лексическое представление с использованием шестнадцатеричной или восьмеричной записи.

Шестнадцатеричное представление начинается с необязательного знака (+ или -), за которым следует пара символов 0x, а далее последовательность шестнадцатеричных цифр, в которой могут использоваться символы верхнего или нижнего регистра (a - f, A - F). Восьмеричное представления начинается с необязательного знака (+ или -), за которым следует символ 0 и последовательность восьмеричных цифр (0 - 7).

Отметим, что принятое по умолчанию значение в модуле YANG, которое начинается с нуля (0), интерпретируется как восьмеричное число. В представлении XML все числа считаются десятичными и нули в начале значение допускаются.

Примеры:

```
// пригодные значения
+4711                    // допустимое положительное значение
4711                     // допустимое положительное значение
-123                     // допустимое отрицательное значение
0xf00f                   // допустимое положительное шестнадцатеричное значение
-0xf                     // допустимое отрицательное шестнадцатеричное значение
052                      // допустимое положительное восьмеричное значение

// непригодные значения
- 1                      // недопустимый пробел
```

9.2.2. Каноническая форма

Каноническая форма положительного целого числа не включает знака +. Нули в начале последовательности цифр не допускаются. Нулевое значение представляется одним символом 0.

9.2.3. Ограничения

Все целые числа могут ограничиваться с помощью оператора range (параграф 9.2.4).

9.2.4. Оператор range

Оператор range, является необязательным субоператором для оператора type и принимает в качестве аргумента строку, выражающую диапазон. Субоператор служит для ограничения диапазонов значений целых и десятичных встроенных типов и производных от них.

Диапазон задаётся явным значением или включительной нижней границей, за которой следуют два символа точки (..) и включительное значение верхней границы. Возможно задание множества значений или диапазонов, разделённых символом |. При задании множеств диапазонов они **должны** быть не пересекающимися и **должны** указываться в порядке роста значений. Если ограничение range применяется к типу, который уже имеет такие ограничения, новое ограничение **должно** соответствовать имеющемуся или быть более сильным (т. е. повышающим нижнюю или/и снижающим верхнюю границу, удаляющим явно заданные значения или расщепляющим диапазон на поддиапазоны с зазорами между ними). Каждое явное значение и граница диапазона в выражении range **должно** соответствовать ограничиваемому типу или быть одним из специальных значений min или max (минимальное и максимальное допустимые значения для типа, соответственно).

Синтаксис выражения range формально определяется правилом range-arg в разделе 12.

9.2.4.1. Субоператоры для range

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	0..1

9.2.5. Пример использования

```
typedef my-base-int32-type {
    type int32 {
        range "1..4 | 10..20";
    }
}
typedef my-type1 {
  type my-base-int32-type {
    // допустимое ограничение диапазона
    range "11..max"; // 11..20
  }
}
typedef my-type2 {
  type my-base-int32-type {
    // недопустимое ограничение диапазона
    range "11..100";
  }
}
```

9.3. Встроенный тип decimal64

Встроенный тип decimal64 представляет подмножество действительных чисел, которые могут быть выражены последовательностями десятичных цифр. Пространство значений decimal64 представляет собой подмножество значений, которые могут быть получены путём умножения 64-битового целого числа со знаком на отрицательную степень числа 10 (т. е. значения вида i * 10-n, где i - значение типа integer64, n - целое число от 1 до 18, включительно).

9.3.1. Лексическое представление

Значение decimal64 лексически представляется необязательным знаком (+ или -), за которым следует набор десятичных цифр, а за ним может следовать символ точки (.) в качестве разделителя целой и дробной части и цепочка десятичных цифр дробной части. Если знак не указан, по умолчанию предполагается +.

9.3.2. Каноническая форма

Каноническая форма положительного значения decimal64 не включает знак +. Десятичная точка обязательна. Нули в начале и в конце (дробной части) числа использовать запрещено с учётом того, что перед десятичной точкой и после неё должна присутствовать хотя бы одна цифра. Нулевое значение представляется в виде 0.0.

9.3.3. Ограничения

Тип decimal64 может быть ограничен с помощью оператора range (параграф 9.2.4).

9.3.4. Оператор fraction-digits

Оператор fraction-digits, который служит субоператором для type, **должен** присутствовать для типа decimal64. Он принимает в качестве аргумента целое число от 1 до 18, включительно. Это значение управляет разностью между смежными значениями decimal64 путём ограничения пространства значений, которое выражается как i * 10-n, n - число цифр дробной части.

В таблице представлены минимальные и максимальные значения для каждого размера дробной части.

Цифр после запятой	*Минимум*	*Максимум*
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. Пример использования

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

9.4. Встроенный тип string

Встроенный тип string представляет в языке YANG текстовые строки, предназначенные для человека. Допустимо использование символов кодировок Unicode и ISO/IEC 10646 [ISO.10646], включая табуляцию, возврат каретки и перевод строки.

```
;; любой символ Unicode за исключением суррогатных блоков FFFE и FFFF.
string = *char
char = %x9 / %xA / %xD / %x20-D7FF / %xE000-FFFD /
       %x10000-10FFFF
```

9.4.1. Лексическое представление

Значение string лексически представляется символьными данными в документах экземпляров XML.

9.4.2. Каноническая форма

Каноническая форма совпадает с лексическим представлением. Нормализация Unicode для string не применяется.

9.4.3. Ограничения

Строка (string) может быть ограничена операторами length (параграф 9.4.4) и pattern (параграф 9.4.6).

9.4.4. Оператор length

Оператор length, который является необязательным субоператором для type, принимает в качестве аргумента строку с выражением размера. Оператор служит для ограничения встроенного типа string, а также производных от него.

Оператор length ограничивает число символов Unicode в строке.

Диапазон размеров задаётся явным значением или включительной нижней границей, за которой следуют два символа точки (..) и включительное значение верхней границы. Возможно задание множества значений или диапазонов, разделённых символом |. При задании множеств диапазонов они **должны** быть не пересекающимися и **должны**

указываться в порядке роста значений. Использование отрицательных значений **недопустимо**. Если ограничение диапазона применяется к типу, который уже имеет такие ограничения, новое ограничение **должно** соответствовать имеющемуся или быть более сильным (т. е. повышающим нижнюю или/и снижающим верхнюю границу, удаляющим явно заданные значения или расщепляющим диапазон на поддиапазоны с зазорами между ними). Значение размера представляет собой неотрицательное целое число или одно из специальных значений min или max (минимальное и максимальное допустимые значения для типа, соответственно). От реализаций не требуется поддержка размеров больше 18446744073709551615.

Синтаксис выражения length формально описывается правилом length-arg в разделе 12.

9.4.4.1. Субоператоры для length

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	0..1

9.4.5. Пример использования

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

type my-base-str-type {
    // действительное ограничение размера
    length "11 | 42..max"; // 11 | 42..255
}

type my-base-str-type {
    // недопустимое ограничение размера
    length "1..999";
}
```

9.4.6. Оператор pattern

Оператор pattern, который является необязательным субоператором для type, принимает в качестве аргумента строку регулярного выражения, в соответствии с определением [XSD-TYPES]. Он служит для ограничения встроенного типа string и производных от него типов, значениями, которые соответствуют шаблону pattern.

Если тип (type) имеет множество операторов pattern, выражения объединяются логической операцией AND (И), т. е. строка должна соответствовать всем выражениям.

Если шаблонные ограничения применяются к типу, который уже имеет ограничения по шаблонам, значения должны соответствовать всем шаблонам базового типа в дополнение к новым шаблонам.

9.4.6.1. Субоператоры для pattern

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.19.4	0..1

9.4.7. Пример использования

Для показанного ниже типа

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

следующие строки соответствуют

```
AB          // действительно
9A00        // действительно
```

а следующие строки не соответствуют

```
00ABAB      // слишком длинная
xx00        // непригодные символы
```

9.5. Встроенный тип boolean

Встроенный тип boolean представляет логические значения.

9.5.1. Лексическое представление

Лексическое представление типа boolean имеет два значения true и false, которые **должны** указываться символами нижнего регистра.

9.5.2. Каноническая форма

Каноническая форма совпадает с лексическим представлением.

9.5.3. Ограничения

Тип boolean не может быть ограничен.

9.6. Встроенный тип enumeration

Встроенный тип enumeration представляет значения из набора заданных имён.

9.6.1. Лексическое представление

Лексическим представлением значения enumeration является строка назначенного имени.

9.6.2. Каноническая форма

Канонической формой является строка назначенного имени.

9.6.3. Ограничения

Тип enumeration может быть ограничен с помощью одного или нескольких операторов enum (параграф 9.6.4), которые указывают подмножество значений базового типа.

9.6.4. Оператор enum

Оператор enum, который является субоператором для type, **должен** присутствовать для типа enumeration. Он может использоваться несколько раз для задания каждого назначенного имени типа enumeration. Оператор принимает в качестве аргумента строку назначенного имени. **Недопустимо** указание пустых строк (нулевой размер), а также **недопустимо** добавление пробельных символов в начале или в конце назначенного имени. **Следует** избегать использования в именах управляющих кодов Unicode.

Оператор может включать необязательный блок субоператоров с дополнительной информацией.

Назначенные имена в enumeration **должны** быть уникальными.

При ограничении типа enumeration набор имён нового типа **должен** быть подмножеством базового набора назначенных имён. **Недопустимо** изменять эти назначенные имена.

9.6.4.1. Субоператоры для enum

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
reference	7.19.4	0..1
status	7.19.2	0..1
value	9.6.4.2	0..1

9.6.4.2. Оператор value

Необязательный оператор value служит для связывания целочисленного значения с назначенным именем для enum. Это **должно** быть целое число из диапазона от -2147483648 до 2147483647 и значение **должно** быть уникальным в рамках типа enumeration. Значение не используется в YANG и XML, но передаётся для удобства разработчиков.

Если значение не задано, оно будет назначаться автоматически. В таких случаях при первом определении будет назначаться значение (0), а противном случае - значение, превышающее на единицу максимальное из использованных значений enum.

Если текущее максимальное значение составляет 2147483647, значение enum **должно** быть задано для субоператоров enum, следующих за текущим максимальным значением.

9.6.5. Пример использования

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

Лексическое представление листа myenum со значением seven будет иметь вид

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

9.7. Встроенный тип bits

Встроенный тип bits представляет набор битов. Т. е. значение bits служит набором флагов, указанных целочисленными значениями позиции с отсчётом от 0. Каждому биту назначено имя.

9.7.1. Ограничения

Тип bits может быть ограничен с помощью оператора bit (параграф 9.7.4).

9.7.2. Лексическое представление

Лексическое представление типа bits является списком разделённых пробелами назначенных имён флагов bits. Пустая строка говорит об отсутствии установленных флагов.

9.7.3. Каноническая форма

В канонической форме значения битов разделяются одним символом пробела и упорядочиваются по их позициям (см. параграф 9.7.4.2).

9.7.4. Оператор bit

Оператор bit, который служит субоператором для type, **должен** присутствовать для типа bits. Он используется многократно для каждого именованного бита типа bits. Оператор принимает в качестве аргумента строку с назначенным именем бита, за которой следует блок субоператоров с дополнительной информацией. Для назначенных имён используется такой же синтаксис, как для идентификаторов (см. параграф 6.2).

Назначенные имена в типе bits **должны** быть уникальными.

9.7.4.1. Субоператоры для bit

Субоператор	*Параграф*	*Число элементов*
description	7.19.3	0..1
position	9.7.4.2	0..1
reference	7.19.4	0..1
status	7.19.2	0..1

9.7.4.2. Оператор position

Необязательный оператор position принимает в качестве аргумента неотрицательное целое число, которое задаёт позицию бита в предполагаемом битовом поле. Значение position **должно** лежать в диапазоне от 0 до 4294967295 и **должно** быть уникальным в данном типе bits. Значение не используется в YANG и NETCONF, но передаётся для удобства разработчиков.

Если позиция бита не задана, она назначается автоматически. Если субоператор bit задан в первый раз, присваивается значение 0, в противном случае присваивается значение на 1 больше текущего максимального значения битовой позиции.

Если текущее максимальное значение составляет 4294967295, битовая позиция **должна** быть задана для субоператоров bit, следующих за текущим максимальным значением.

9.7.5. Пример использования

Для приведённого ниже leaf

```
leaf mybits {
    type bits {
        bit disable-nagle {
            position 0;
        }
        bit auto-sense-speed {
            position 1;
        }
        bit 10-Mb-only {
            position 2;
        }
    }
    default "auto-sense-speed";
}
```

Лексическое представление листа с битовыми полями disable-nagle и 10-mb-only будет иметь вид

```
<mybits>disable-nagle 10-Mb-only</mybits>
```

9.8. Встроенный тип binary

Встроенный тип binary представляет любые двоичные данные, т. е. последовательность октетов.

9.8.1. Ограничения

Тип binary может быть ограничен с помощью оператора length (параграф 9.4.4). Размер значения типа binary указывает число октетов.

9.8.2. Лексическое представление

Значения binary представляются с использованием схемы кодирования base64 (см. раздел 4 в [RFC4648]).

9.8.3. Каноническая форма

Каноническая форма значения binary следует правилам кодирования base64 [RFC4648].

9.9. Встроенный тип leafref

Тип leafref служит для ссылки на конкретный экземпляр leaf в дереве данных. Субоператор path (параграф 9.9.2) служит для выбора набора экземпляров leaf и пространством значений leafref является набор значений экземпляров leaf.

Если свойство require-instance (параграф 9.9.3) имеет значение true, **должен** существовать узел в дереве данных, или узел, использующий принятое по умолчанию значение (см. параграфы 7.6.1 и 7.7.2), для указанного узла leaf или leaf-list в дереве схемы с таким же значением, как leafref в действительном дереве данных.

Если лист с типом leafref представляет данные конфигурации, указанный ссылкой узел также **должен** представлять конфигурацию. Такой лист задаёт ограничения для пригодных данных. Все узлы leafref **должны** указывать на имеющиеся экземпляры leaf или листья, с принятыми по умолчанию значениями (см. параграф 7.6.1), чтобы данные были действительны. Это ограничение применяется в соответствии с правилами раздела 8.

Недопустимы замкнутые цепочки ссылок leafrefs.

Если лист, на который указывает leafref, зависит от одной или нескольких функций (см. параграф 7.20.2), узел leaf с типом leafref **должен** зависеть от того же набора функций.

9.9.1. Ограничения

Тип leafref не может быть ограничен.

9.9.2. Оператор path

Оператор path, являющийся субоператором для type, **должен** присутствовать для типа leafref. Оператор принимает в качестве аргумента строку, которая **должна** указывать на узел leaf или leaf-list.

Синтаксис аргумента path является подмножеством сокращённого синтаксиса XPath. Предикаты служат лишь для ограничения значений узлов ключей для элементов списка. Каждый предикат содержит в точности одну проверку равенства на ключ и **может** присутствовать множество смежных предикатов, если список имеет множество ключей. Синтаксис формально определён правилом path-arg в разделе 12.

Предикаты используются лишь в тех случаях, когда требуется более одной ссылки на ключ для однозначной идентификации экземпляра листа. Это происходит, если список имеет имеет множество ключей или нужна ссылка на лист, отличающаяся от ключа в списке. В таких случаях обычно задаётся множество leafref, а предикаты служат для их связывания воедино.

Выражение path даёт набор узлов (возможно пустой). Если свойство require-instance имеет значение true, набор узлов **должен** быть непустым.

Выражение XPath в операторе path концептуально оценивается в указанном ниже варианте контекста в дополнение к приведённому в параграфе 6.4.1 определению.

- Узлом контекста является узел в дереве данных для которого определён оператор path.

Доступное дерево зависит от узла контекста.

- Если узел контекста представляет данные конфигурации, дерево будет данными в хранилище NETCONF, где существует узел контекста. Корневой узел XPath имеет в качестве потомков все узлы конфигурационных данных верхнего уровня во всех модулях.
- В остальных случаях дерево будет данными состояния на устройстве и в хранилище <running/>. Корневой узел XPath имеет в качестве потомков все узлы данных верхнего уровня во всех модулях.

9.9.3. Лексическое представление

Значение leafref лексически представляется так же, как представляет своё значение узел leaf, на который указывает ссылка.

9.9.4. Каноническая форма

Каноническая форма leafref совпадает с канонической формой листа (leaf) на который указывает ссылка.

9.9.5. Пример использования

Для приведённого ниже списка

```
list interface {
    key "name";
    leaf name {
        type string;
    }
    leaf admin-status {
        type admin-status;
    }
    list address {
        key "ip";
        leaf ip {
            type yang:ip-address;
        }
    }
}
```

leafref указывает на существующий интерфейс

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

Ниже представлен соответствующий фрагмент XML.

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

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

Приведённый ниже оператор leafrefs указывает на существующий адрес интерфейса.

```
container default-address {
```

```
        leaf ifname {
            type leafref {
                path "../../interface/name";
            }
        }
        leaf address {
            type leafref {
                path "../../interface[name = current()/../ifname]"
                   + "/address/ip";
            }
        }
    }
```

Соответствующий фрагмент XML имеет вид

```
<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>
```

Приведённый ниже список использует leafref для одного из своих ключей. Это похоже на внешний ключ в реляционной базе данных.

```
list packet-filter {
    key "if-name filter-id";
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf filter-id {
        type uint32;
    }
    ...
}
```

Соответствующий фрагмент XML имеет вид

```
<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>
```

Приведённое ниже уведомление определяет два leafrefs для ссылки на существующий лист admin-status

```
notification link-failure {
    leaf if-name {
        type leafref {
            path "/interface/name";
        }
    }
    leaf admin-status {
        type leafref {
            path
              "/interface[name = current()/../if-name]"
            + "/admin-status";
```

```
            }
        }
    }
```

Ниже приведён пример соответствующего уведомления XML.

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-04-01T00:01:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>eth0</if-name>
    <admin-status>up</admin-status>
  </link-failure>
</notification>
```

9.10. Встроенный тип identityref

Встроенный тип identityref служит для ссылок на существующие отождествления (см. параграф 7.16).

9.10.1. Ограничения

Тип identityref не может быть ограничен.

9.10.2. Оператор base для identityref

Оператор base, который является субоператором для type, **должен** присутствовать хотя бы однократно для типа identityref. Аргументом служит имя отождествления, определённое оператором identity. Если в имени отождествления имеется префикс, это говорит о том, что отождествление определено в модуле, который импортирован с этим префиксом. В остальных случаях отождествление с совпадающим именем **должно** быть определено в текущем модуле или включённом в него субмодуле.

Допустимыми значениями для identityref являются любые отождествления, произведённые из базового отождествления identityref. На конкретном сервере пригодные значения могут быть дополнительно ограничены набором отождествлений, определённых в модулях, которые поддерживаются сервером.

9.10.3. Лексическое представление

Тип identityref лексически представляется как полное имя указанной ссылки в соответствии с [XML-NAMES]. Если префикс отсутствует, пространством имён identityref будет принятое по умолчанию пространство имён для элемента, содержащего значение identityref.

Когда identityref присваивается принятое по умолчанию значение с использованием оператора default, имя отождествления в принятом по умолчанию значении **может** включать префикс. Если префикс присутствует в имени отождествления, он указывает на отождествление, определённое в модуле, который был импортирован с этим префиксом, или префикс текущего модуля, если префикс указывает текущий модуль или один из его субмодулей. В остальных случаях отождествление с совпадающим именем **должно** быть определено в текущем модуле или включённом субмодуле.

9.10.4. Каноническая форма

Поскольку лексическая форма зависит от контекста XML, в котором это значение имеет место, этот тип не имеет канонической формы.

9.10.5. Пример использования

С определениями отождествлений из параграфа 7.16.3 и приведённым ниже модулем

```
module my-crypto {

    namespace "http://example.com/my-crypto";
    prefix mc;

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

    identity aes {
        base "crypto:crypto-alg";
    }

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

Пример кодирования листа crypto для отождествления des3, определённого в модуле des будет иметь вид

```
<crypto xmlns:des="http://example.com/des">des:des3</crypto>
```

Все префиксы, использованные в представлении, являются локальными для каждого экземпляра представления. Это значит, что один и тот же identityref может по разному кодироваться разными реализациями. Ниже представлен другой пример кодирования того же листа.

```
<crypto xmlns:x="http://example.com/des">x:des3</crypto>
```

Если значение листа crypto вместо модуля aes определено в модуле my-crypto, кодирование может иметь вид

```
<crypto xmlns:mc="http://example.com/my-crypto">mc:aes</crypto>
```

или с использованием принятого по умолчанию пространства имён

```
<crypto>aes</crypto>
```

9.11. Встроенный тип empty

Встроенный тип empty представляет лист, не имеющий какого-либо значения, он лишь даёт информацию о наличии или отсутствии.

Тип empty не может иметь принятого по умолчанию значения.

9.11.1. Ограничения

Тип empty не может быть ограничен.

9.11.2. Лексическое представление

Не применимо.

9.11.3. Каноническая форма

Не применима.

9.11.4. Пример использования

Для показанного ниже листа

```
leaf enable-qos {
    type empty;
}
```

примером пригодного представления существования листа может быть

```
<enable-qos/>
```

9.12. Встроенный тип union

Встроенный тип union представляет значение, которое соответствует типу одного из членов объединения.

Когда указан тип union, **должен** присутствовать оператор type (параграф 7.4). Он используется многократно для задания типа каждого члена объединения. Оператор принимает один аргумент, являющийся строкой с именам типа.

В объединение могут входить встроенные и производные типы, однако **недопустимо** использование встроенных типов empty и leafref.

При проверке пригодности строки, представляющей тип данных union проверка осуществляется для каждого типа, являющегося членом объединения, в порядке, заданном оператором type, пока не будет найдено совпадение.

Любое значение, принятое по умолчанию, или свойство units, определённое в типах членов, не наследуется типом union.

Например,

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

9.12.1. Ограничения

Тип union не может иметь ограничений. Однако каждый из входящих в объединение типов может иметь ограничения, определённые в разделе 9.

9.12.2. Лексическое представление

Лексическим представлением типа union является значение, которое соответствует представлению одного (любого) из входящих в объединение типов.

9.12.3. Каноническая форма

Каноническая форма значения union совпадает с канонической формой значения типа, входящего в объединение.

9.13. Встроенный тип instance-identifier

Встроенный тип instance-identifier служит для однозначной идентификации конкретного экземпляра узла в дереве данных.

Синтаксис для instance-identifier является подмножеством сокращённого синтаксиса XPath и формально определён правилом instance-identifier в разделе 12. Тип применяется для однозначной идентификации узлов в дереве данных. Предикаты применяются только для задания значений ключевых узлов элементов списка, значений элементов leaf-list или позиционных индексов в списках без ключей. Для идентификации элементов списка с ключами каждый предикат состоит из одной проверки равенства на ключ и каждый ключ **должен** иметь соответствующие предикат.

Если лист типа instance-identifier представляет данные конфигурации и свойство require-instance (параграф 9.13.2) имеет значение true, узел, на который он ссылается, также **должен** представлять конфигурацию. Такой лист ограничивает пригодные данные. Узлы всех таких листьев **должны** указывать существующие узлы или узлы leaf, использующие принятые по умолчанию значения (см. параграфы 7.6.1), для того, чтобы данные были пригодны. Это ограничение применяется в соответствии с правилами раздела 8.

Выражение instance-identifier XPath концептуально проверяется в контексте корневого узла доступного дерева в дополнение к определению параграфа 6.4.1.

- Узлом контекста является корневой узел в доступном дереве.

Доступное дерево зависит от листа с типом instance-identifier.

- Если узел контекста представляет данные конфигурации, дерево будет данными в хранилище NETCONF, где существует узел контекста. Корневой узел XPath имеет в качестве потомков все узлы конфигурационных данных верхнего уровня во всех модулях.
- В остальных случаях дерево будет данными состояния на устройстве и в хранилище <running/>. Корневой узел XPath имеет в качестве потомков все узлы данных верхнего уровня во всех модулях.

9.13.1. Ограничения

Тип instance-identifier может быть ограничен с помощью оператора require-instance (параграф 9.13.2).

9.13.2. Оператор require-instance

Оператор require-instance, который является субоператором для type, **может** присутствовать для типа instance-identifier. Он принимает в качестве аргумента значение true или false. Если оператор отсутствует, по умолчанию принимается значение true.

Если require-instance имеет значение true, это означает, что экземпляр, на который указывается ссылка, **должен** существовать, чтобы данные были действительны. Ограничения применяются в соответствии с правилами раздела 8.

Если require-instance имеет значение false, это значит, что экземпляр, на который указывает ссылка, **может** существовать в действительных данных.

9.13.2. Лексическое представление

Значение instance-identifier лексически представляется в форме строки. Все имена узлов в значении instance-identifier **должны** быть полными с явными префиксами пространства имён и эти префиксы **должны** быть заявлены в области действия пространства имён XML в элементе XML для instance-identifier.

Все префиксы, используемые в представлении, являются локальными для каждого экземпляра кодирования. Это означает, что instance-identifier может по разному кодироваться разными реализациями.

9.13.3. Каноническая форма

Поскольку лексическая форма зависит от контекста XML, в котором значение имеет место, для этого типа нет канонической формы.

9.13.4. Пример использования

Ниже приведены примеры использования идентификаторов экземпляра.

```
/* instance-identifier для container */
/ex:system/ex:services/ex:ssh

/* instance-identifier для leaf */
/ex:system/ex:services/ex:ssh/ex:port

/* instance-identifier для записи list */
/ex:system/ex:user[ex:name='fred']

/* instance-identifier для leaf в записи list */
/ex:system/ex:user[ex:name='fred']/ex:type

/* instance-identifier для записи list с двумя ключами */
/ex:system/ex:server[ex:ip='192.0.2.1'][ex:port='80']

/* instance-identifier для записи leaf-list */
/ex:system/ex:services/ex:ssh/ex:cipher[.='blowfish-cbc']

/* instance-identifier для записи list без ключей */
/ex:stats/ex:port[3]
```

10. Обновление модулей

По мере использования модуля может возникнуть потребность в его обновлении. Однако обновление опубликованных модулей не допускается, если это может вызвать проблемы взаимодействия между клиентами, использующими исходную спецификацию, и сервером, использующим обновлённую спецификацию.

При публикации обновления **должен** использоваться новый оператор revision (параграф 7.1.9), помещаемый перед имеющимися операторами revision. Если в модуле нет операторов revision, такой оператор **должен** просто добавляться для обозначения нового выпуска. Кроме того, **должны** быть внесены все требуемые изменения в операторы metadata, включая organization и contact (см. параграфы 7.1.7 и 7.1.8).

Отметим, что содержащиеся в модуле определения доступны для импорта любому другому модулю путём указания имени данного модуля в операторе import. По этой причине менять имя модуля **недопустимо**. Кроме того, **недопустимо** менять оператор namespace, поскольку все элементы XML привязаны к пространству имён.

Устаревшие определения **недопустимо** удалять из опубликованных модулей, поскольку их идентификаторы ещё могут применяться другими модулями.

Определение в опубликованном модуле может быть пересмотрено любым из приведённых ниже способов.

- Для типа enumeration могут быть добавлены новые элементы enum с сохранением значения для существующих элементов.
- Для типа bits могут быть добавлены новые флаги с сохранением позиций имеющихся флагов.
- Операторы range, length или pattern могут расширять пространство разрешённых значений.
- Может быть добавлен оператор default для листа (leaf), который не имел принятого по умолчанию значения (напрямую или опосредовано через его тип).
- Может быть добавлен оператор units.
- Может быть добавлен или изменён оператор reference.
- Оператор must может быть удалён или его ограничения ослаблены.
- Оператор when может быть удалён или его ограничения ослаблены.
- Оператор mandatory может быть удалён или значение true заменено на false.
- Оператор min-elements может быть удалён или изменён, чтобы требовать меньше элементов.
- Оператор max-elements может быть удалён или изменён, чтобы разрешить больше элементов.
- Оператор description может быть добавлен или уточнён без изменения семантики определения.
- Могут быть добавлены новые операторы typedef, grouping, rpc, notification, extension, feature и identity.
- Могут быть добавлены операторы определения данных, если они не добавляют обязательных узлов (параграф 3.1) к имеющимся узлам или на верхнем уровне модуля или субмодуля, а также не создают для тех условных зависимостей от новой функции (т. е. if-feature с указанием новой функции).
- Могут быть добавлены варианты case.
- Узел, представляющий данные состояния, может быть изменён для представления данных конфигурации, если он не является обязательным (параграф 3.1).
- Оператор if-feature может быть удалён, если его узел не является обязательным (параграф 3.1).
- Оператор status может быть добавлен или значение его изменено с current на deprecated или obsolete, а также с deprecated на obsolete.
- Оператор type может быть заменён другим оператором type, который не меняет синтаксис и семантику типа. Например, определение типа inline может быть заменено на typedef, но тип int8 нельзя заменить на int16, поскольку это меняет синтаксис.
- Любой набор узлов определения данных может быть заменён другим набором синтаксически и семантически эквивалентных узлов. Например, набор узлов может быть заменён оператором uses группировки с тем же набором листьев.
- Модуль может быть разделен на несколько субмодулей или часть субмодулей может быть удалена, если определения модуля не изменятся (за исключением разрешённых здесь изменений).
- Оператор prefix может быть изменён при условии корректировки всех локальных применений этого префикса.

В остальных случаях при изменении семантики прежнего определения (т. е. внесении изменений, выходящих за рамки перечисленного выше) **должно** создаваться новое определение с новым идентификатором.

В операторах, включающих какие-либо операторы определения данных в качестве своих субоператоров, **недопустимо** менять порядок этих субоператоров с определениями.

11. YIN

Модуль YANG может быть преобразован в другой основанный на XML формат, называемый YIN. Преобразованный модуль называется модулем YIN. В этом разделе описаны правила прямого и обратного преобразования между двумя форматами.

Форматы YANG и YIN содержат эквивалентную информацию, но используют разные обозначения. Нотация YIN позволяет разработчикам представлять модули данных YANG в формате XML, что даёт возможность использования богатого набора инструментов XML для фильтрации и проверки данных, автоматизированной генерации кода и других задач. Могут применяться инструменты типа XSLT или валидаторов XML.

Преобразование между YANG и YIN не меняет информационного содержимого модели. Комментарии и пробельные символы не сохраняются.

11.1. Формальное определение YIN

Между ключевыми словами YANG и элементами YIN существует взаимно-однозначное соответствие. Локальное имя элемента YIN идентично соответствующему ключевому слову YANG. Это означает, в частности, что корневым элементом документа YIN всегда будет <module> или <submodule>.

Элементы YIN, соответствующие ключевым словам YANG, относятся к пространству имён, идентификатор URI которого имеет значение urn:ietf:params:xml:ns:yang:yin:1.

Элементы YIN, соответствующие ключевым словам расширений, относятся к пространству имён модуля YANG, в котором ключевое слово было объявлено с помощью оператора extension.

Имена всех элементов YIN **должны** быть надлежащим образом определены в их пространствах имён (см. выше) с использованием стандартных механизмов [XML-NAMES], т. е. атрибутов xmlns и xmlns:xxx.

Аргумент оператора YANG представляется в YIN атрибутом XML или субэлементом элемента keyword. Отображения для ключевых слов YANG приведены в таблице 1. Для расширений отображение аргумента задаётся оператором extension (см. параграф 7.17) и приведёнными ниже правилами.

- Если аргумент представлен атрибутом, этот атрибут не имеет пространства имён.
- Если аргумент представлен элементом, он определяется тем же пространством имён, что и его родительский элемент keyword.
- Если аргумент представлен как элемент, он **должен** быть первым потомком элемента keyword.

Субоператоры операторов YANG представляются как (дополнительные) потомки элемента keyword, а их относительный порядок **должен** совпадать с порядком субоператоров в YANG.

Комментарии YANG **могут** преобразовываться в комментарии XML.

Таблица 1. Отображение аргументов операторов YANG.

Ключевое слово	***Имя аргумента***	***yin-element***
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
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

11.1.1. Пример использования

Приведённый ниже модуль YANG

```
module acme-foo {
    namespace "http://acme.example.com/foo";
    prefix "acfoo";

    import my-extensions {
        prefix "myext";
    }

    list interface {
        key "name";
        leaf name {
            type string;
        }

        leaf mtu {
            type uint32;
            description "MTU для интерфейса.";
            myext:c-define "MY_MTU";
        }
    }
}
```

где расширение c-define определено в параграфе 7.17.3, преобразуется в YIN вида

```
<module name="acme-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:acfoo="http://acme.example.com/foo"
        xmlns:myext="http://example.com/my-extensions">

  <namespace uri="http://acme.example.com/foo"/>
  <prefix value="acfoo"/>

  <import module="my-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>MTU для интерфейса.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

12. Грамматика ABNF для YANG

В YANG почти все операторы являются неупорядоченными. Грамматика ABNF [RFC5234] определяет канонический порядок. Для улучшения читаемости модулей **рекомендуется** размещать предложения (clause) в этом порядке.

В грамматике ABNF неупорядоченные операторы отмечены комментариями. Предполагается, что сканер заменит комментарий YANG одиночным символом пробела.

```
<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 = ;; эти операторы могут указываться в любом порядке
                      [yang-version-stmt stmtsep]
                       namespace-stmt stmtsep
                       prefix-stmt stmtsep

submodule-header-stmts =
                      ;; эти операторы могут указываться в любом порядке
                      [yang-version-stmt stmtsep]
                       belongs-to-stmt stmtsep

meta-stmts          = ;; эти операторы могут указываться в любом порядке
                      [organization-stmt stmtsep]
                      [contact-stmt stmtsep]
                      [description-stmt stmtsep]
                      [reference-stmt stmtsep]

linkage-stmts       = ;; эти операторы могут указываться в любом порядке
                      *(import-stmt stmtsep)
                      *(include-stmt stmtsep)

revision-stmts      = *(revision-stmt stmtsep)

body-stmts          = *((extension-stmt /
                         feature-stmt /
                         identity-stmt /
                         typedef-stmt /
                         grouping-stmt /
                         data-def-stmt /
                         augment-stmt /
                         rpc-stmt /
                         notification-stmt /
                         deviation-stmt) stmtsep)

data-def-stmt       = container-stmt /
                      leaf-stmt /
                      leaf-list-stmt /
                      list-stmt /
                      choice-stmt /
                      anyxml-stmt /
                      uses-stmt

yang-version-stmt   = yang-version-keyword sep yang-version-arg-str
                      optsep stmtend

yang-version-arg-str = < a string that matches the rule
                         yang-version-arg >

yang-version-arg    = "1"

import-stmt         = import-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          prefix-stmt stmtsep
                          [revision-date-stmt stmtsep]
                      "}"

include-stmt        = include-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           [revision-date-stmt stmtsep]
                       "}")

namespace-stmt      = namespace-keyword sep uri-str optsep stmtend

uri-str             = < a string that matches the rule
                        URI in RFC 3986 >

prefix-stmt         = prefix-keyword sep prefix-arg-str
                      optsep stmtend

belongs-to-stmt     = belongs-to-keyword sep identifier-arg-str
                      optsep
                      "{" stmtsep
                          prefix-stmt stmtsep
                      "}"

organization-stmt   = organization-keyword sep string
                      optsep stmtend

contact-stmt        = contact-keyword sep string optsep stmtend

description-stmt    = description-keyword sep string optsep
                      stmtend
```

```
reference-stmt      = reference-keyword sep string optsep stmtend

units-stmt          = units-keyword sep string optsep stmtend

revision-stmt       = revision-keyword sep revision-date optsep
                      (";" /
                       "{" stmtsep
                           [description-stmt stmtsep]
                           [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
                           ;; эти операторы могут указываться в любом порядке
                           [argument-stmt stmtsep]
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [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
                           ;; эти операторы могут указываться в любом порядке
                           [base-stmt stmtsep]
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                       "}")

base-stmt           = base-keyword sep identifier-ref-arg-str
                      optsep stmtend

feature-stmt        = feature-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут указываться в любом порядке
                           *(if-feature-stmt stmtsep)
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                       "}")

if-feature-stmt     = if-feature-keyword sep identifier-ref-arg-str
                      optsep stmtend

typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут указываться в любом порядке
                          type-stmt stmtsep
                          [units-stmt stmtsep]
                          [default-stmt stmtsep]
                          [status-stmt stmtsep]
                          [description-stmt stmtsep]
                          [reference-stmt stmtsep]
                       "}"

type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                      (";" /
                       "{" stmtsep
                           type-body-stmts
                       "}")

type-body-stmts     = numerical-restrictions /
                      decimal64-specification /
                      string-restrictions /
```

```
                        enum-specification /
                        leafref-specification /
                        identityref-specification /
                        instance-identifier-specification /
                        bits-specification /
                        union-specification

numerical-restrictions = range-stmt stmtsep

range-stmt           = range-keyword sep range-arg-str optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут указываться в любом порядке
                            [error-message-stmt stmtsep]
                            [error-app-tag-stmt stmtsep]
                            [description-stmt stmtsep]
                            [reference-stmt stmtsep]
                         "}")

decimal64-specification = fraction-digits-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 = ;; эти операторы могут указываться в любом порядке
                      [length-stmt stmtsep]
                      *(pattern-stmt stmtsep)

length-stmt          = length-keyword sep length-arg-str optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут указываться в любом порядке
                            [error-message-stmt stmtsep]
                            [error-app-tag-stmt stmtsep]
                            [description-stmt stmtsep]
                            [reference-stmt stmtsep]
                         "}")

pattern-stmt         = pattern-keyword sep string optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут указываться в любом порядке
                            [error-message-stmt stmtsep]
                            [error-app-tag-stmt stmtsep]
                            [description-stmt stmtsep]
                            [reference-stmt stmtsep]
                         "}")

default-stmt         = default-keyword sep string stmtend

enum-specification   = 1*(enum-stmt stmtsep)

enum-stmt            = enum-keyword sep string optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут указываться в любом порядке
                            [value-stmt stmtsep]
                            [status-stmt stmtsep]
                            [description-stmt stmtsep]
                            [reference-stmt stmtsep]
                         "}")

leafref-specification =
                      ;; эти операторы могут указываться в любом порядке
                      path-stmt stmtsep
                      [require-instance-stmt stmtsep]

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 stmtsep]

identityref-specification =
                      base-stmt stmtsep

union-specification = 1*(type-stmt stmtsep)

bits-specification  = 1*(bit-stmt stmtsep)

bit-stmt            = bit-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут указываться в любом порядке
                           [position-stmt stmtsep]
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [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
                           ;; эти операторы могут указываться в любом порядке
                           [error-message-stmt stmtsep]
                           [error-app-tag-stmt stmtsep]
                           [description-stmt stmtsep]
                           [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 stmtend

   grouping-stmt       = grouping-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; эти операторы могут указываться в любом порядке
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              *(data-def-stmt stmtsep)
                          "}")

   container-stmt      = container-keyword sep identifier-arg-str optsep
                         (";" /
                          "{" stmtsep
                              ;; эти операторы могут указываться в любом порядке
                              [when-stmt stmtsep]
                              *(if-feature-stmt stmtsep)
                              *(must-stmt stmtsep)
                              [presence-stmt stmtsep]
                              [config-stmt stmtsep]
                              [status-stmt stmtsep]
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              *(data-def-stmt stmtsep)
                          "}")

   leaf-stmt           = leaf-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; эти операторы могут указываться в любом порядке
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             type-stmt stmtsep
                             [units-stmt stmtsep]
                             *(must-stmt stmtsep)
                             [default-stmt stmtsep]
                             [config-stmt stmtsep]
                             [mandatory-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                          "}"

   leaf-list-stmt      = leaf-list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; эти операторы могут указываться в любом порядке
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             type-stmt stmtsep
                             [units-stmt stmtsep]
                             *(must-stmt stmtsep)
                             [config-stmt stmtsep]
                             [min-elements-stmt stmtsep]
                             [max-elements-stmt stmtsep]
                             [ordered-by-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                          "}"

   list-stmt           = list-keyword sep identifier-arg-str optsep
                         "{" stmtsep
                             ;; эти операторы могут указываться в любом порядке
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             *(must-stmt stmtsep)
                             [key-stmt stmtsep]
                             *(unique-stmt stmtsep)
                             [config-stmt stmtsep]
                             [min-elements-stmt stmtsep]
                             [max-elements-stmt stmtsep]
                             [ordered-by-stmt stmtsep]
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                             *((typedef-stmt /
```

```
                            grouping-stmt) stmtsep)
                        1*(data-def-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
                           ;; эти операторы могут указываться в любом порядке
                           [when-stmt stmtsep]
                           *(if-feature-stmt stmtsep)
                           [default-stmt stmtsep]
                           [config-stmt stmtsep]
                           [mandatory-stmt stmtsep]
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                           *((short-case-stmt / case-stmt) stmtsep)
                       "}")

short-case-stmt     = container-stmt /
                      leaf-stmt /
                      leaf-list-stmt /
                      list-stmt /
                      anyxml-stmt

case-stmt           = case-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут указываться в любом порядке
                           [when-stmt stmtsep]
                           *(if-feature-stmt stmtsep)
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                           *(data-def-stmt stmtsep)
                       "}")

anyxml-stmt         = anyxml-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут указываться в любом порядке
                           [when-stmt stmtsep]
                           *(if-feature-stmt stmtsep)
                           *(must-stmt stmtsep)
                           [config-stmt stmtsep]
                           [mandatory-stmt stmtsep]
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                        "}")

uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут указываться в любом порядке
                           [when-stmt stmtsep]
                           *(if-feature-stmt stmtsep)
                           [status-stmt stmtsep]
                           [description-stmt stmtsep]
                           [reference-stmt stmtsep]
                           *(refine-stmt stmtsep)
                           *(uses-augment-stmt stmtsep)
                       "}")

refine-stmt         = refine-keyword sep refine-arg-str optsep
                      (";" /
                       "{" stmtsep
                           (refine-container-stmts /
                            refine-leaf-stmts /
                            refine-leaf-list-stmts /
                            refine-list-stmts /
```

```
                              refine-choice-stmts /
                              refine-case-stmts /
                              refine-anyxml-stmts)
                          "}")

   refine-arg-str      = < a string that matches the rule
                           refine-arg >

   refine-arg          = descendant-schema-nodeid

   refine-container-stmts =
                         ;; эти операторы могут указываться в любом порядке
                         *(must-stmt stmtsep)
                         [presence-stmt stmtsep]
                         [config-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-leaf-stmts   = ;; эти операторы могут указываться в любом порядке
                         *(must-stmt stmtsep)
                         [default-stmt stmtsep]
                         [config-stmt stmtsep]
                         [mandatory-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-leaf-list-stmts =
                         ;; эти операторы могут указываться в любом порядке
                         *(must-stmt stmtsep)
                         [config-stmt stmtsep]
                         [min-elements-stmt stmtsep]
                         [max-elements-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-list-stmts   = ;; эти операторы могут указываться в любом порядке
                         *(must-stmt stmtsep)
                         [config-stmt stmtsep]
                         [min-elements-stmt stmtsep]
                         [max-elements-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-choice-stmts = ;; эти операторы могут указываться в любом порядке
                         [default-stmt stmtsep]
                         [config-stmt stmtsep]
                         [mandatory-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-case-stmts   = ;; эти операторы могут указываться в любом порядке
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   refine-anyxml-stmts = ;; эти операторы могут указываться в любом порядке
                         *(must-stmt stmtsep)
                         [config-stmt stmtsep]
                         [mandatory-stmt stmtsep]
                         [description-stmt stmtsep]
                         [reference-stmt stmtsep]

   uses-augment-stmt   = augment-keyword sep uses-augment-arg-str optsep
                         "{" stmtsep
                             ;; эти операторы могут указываться в любом порядке
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                             1*((data-def-stmt stmtsep) /
                                (case-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
                             ;; эти операторы могут указываться в любом порядке
                             [when-stmt stmtsep]
                             *(if-feature-stmt stmtsep)
                             [status-stmt stmtsep]
                             [description-stmt stmtsep]
                             [reference-stmt stmtsep]
                             1*((data-def-stmt stmtsep) /
                                (case-stmt stmtsep))
```

```
                        "}"

   augment-arg-str      = < a string that matches the rule
                            augment-arg >

   augment-arg          = absolute-schema-nodeid

   unknown-statement    = prefix ":" identifier [sep string] optsep
                          (";" / "{" *unknown-statement2 "}")

   unknown-statement2    = [prefix ":"] identifier [sep string] optsep
                          (";" / "{" *unknown-statement2 "}")

   when-stmt            = when-keyword sep string optsep
                          (";" /
                           "{" stmtsep
                               ;; эти операторы могут указываться в любом порядке
                               [description-stmt stmtsep]
                               [reference-stmt stmtsep]
                            "}")

   rpc-stmt             = rpc-keyword sep identifier-arg-str optsep
                          (";" /
                           "{" stmtsep
                               ;; эти операторы могут указываться в любом порядке
                               *(if-feature-stmt stmtsep)
                               [status-stmt stmtsep]
                               [description-stmt stmtsep]
                               [reference-stmt stmtsep]
                               *((typedef-stmt /
                                  grouping-stmt) stmtsep)
                               [input-stmt stmtsep]
                               [output-stmt stmtsep]
                           "}")

   input-stmt           = input-keyword optsep
                          "{" stmtsep
                              ;; эти операторы могут указываться в любом порядке
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              1*(data-def-stmt stmtsep)
                          "}"

   output-stmt          = output-keyword optsep
                          "{" stmtsep
                              ;; эти операторы могут указываться в любом порядке
                              *((typedef-stmt /
                                 grouping-stmt) stmtsep)
                              1*(data-def-stmt stmtsep)
                          "}"

   notification-stmt    = notification-keyword sep
                          identifier-arg-str optsep
                          (";" /
                           "{" stmtsep
                               ;; эти операторы могут указываться в любом порядке
                               *(if-feature-stmt stmtsep)
                               [status-stmt stmtsep]
                               [description-stmt stmtsep]
                               [reference-stmt stmtsep]
                               *((typedef-stmt /
                                  grouping-stmt) stmtsep)
                               *(data-def-stmt stmtsep)
                           "}")

   deviation-stmt       = deviation-keyword sep
                          deviation-arg-str optsep
                          "{" stmtsep
                              ;; эти операторы могут указываться в любом порядке
                              [description-stmt stmtsep]
                              [reference-stmt stmtsep]
                              (deviate-not-supported-stmt /
                                1*(deviate-add-stmt /
                                   deviate-replace-stmt /
                                   deviate-delete-stmt))
                          "}"

   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 optsep
                          (";" /
```

```
                        "{" stmtsep
                        "}")

deviate-add-stmt    = deviate-keyword sep add-keyword optsep
                      (";" /
                       "{" stmtsep
                           [units-stmt stmtsep]
                           *(must-stmt stmtsep)
                           *(unique-stmt stmtsep)
                           [default-stmt stmtsep]
                           [config-stmt stmtsep]
                           [mandatory-stmt stmtsep]
                           [min-elements-stmt stmtsep]
                           [max-elements-stmt stmtsep]
                       "}")

deviate-delete-stmt = deviate-keyword sep delete-keyword optsep
                      (";" /
                       "{" stmtsep
                           [units-stmt stmtsep]
                           *(must-stmt stmtsep)
                           *(unique-stmt stmtsep)
                           [default-stmt stmtsep]
                       "}")

deviate-replace-stmt = deviate-keyword sep replace-keyword optsep
                      (";" /
                       "{" stmtsep
                           [type-stmt stmtsep]
                           [units-stmt stmtsep]
                           [default-stmt stmtsep]
                           [config-stmt stmtsep]
                           [mandatory-stmt stmtsep]
                           [min-elements-stmt stmtsep]
                           [max-elements-stmt stmtsep]
                       "}")

;; Диапазоны

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

;; Размеры

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

;; Дата

date-arg-str        = < a string that matches the rule
                        date-arg >

date-arg            = 4DIGIT "-" 2DIGIT "-" 2DIGIT

;; Идентификаторы узлов схемы

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

;; Идентификаторы экземпляров

instance-identifier = 1*("/" (node-identifier *predicate))
```

```
   predicate           = "[" *WSP (predicate-expr / pos) *WSP "]"

   predicate-expr      = (node-identifier / ".") *WSP "=" *WSP
                         ((DQUOTE string DQUOTE) /
                          (SQUOTE string SQUOTE))

   pos                 = non-negative-integer-value

   ;; Путь leafref

   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

   ;;; Ключевые слова, использующие базовый синтаксис abnf без учёта регистра

   ;; ключевые слова для операторов
   anyxml-keyword      = 'anyxml'
   argument-keyword    = 'argument'
   augment-keyword     = 'augment'
   base-keyword        = 'base'
   belongs-to-keyword  = 'belongs-to'
   bit-keyword         = 'bit'
   case-keyword        = 'case'
   choice-keyword      = 'choice'
   config-keyword      = 'config'
   contact-keyword     = 'contact'
   container-keyword   = 'container'
   default-keyword     = 'default'
   description-keyword = 'description'
   enum-keyword        = 'enum'
   error-app-tag-keyword = 'error-app-tag'
   error-message-keyword = 'error-message'
   extension-keyword   = 'extension'
   deviation-keyword   = 'deviation'
   deviate-keyword     = 'deviate'
   feature-keyword     = 'feature'
   fraction-digits-keyword = 'fraction-digits'
   grouping-keyword    = 'grouping'
   identity-keyword    = 'identity'
   if-feature-keyword  = 'if-feature'
   import-keyword      = 'import'
   include-keyword     = 'include'
   input-keyword       = 'input'
   key-keyword         = 'key'
   leaf-keyword        = 'leaf'
   leaf-list-keyword   = 'leaf-list'
   length-keyword      = 'length'
   list-keyword        = 'list'
   mandatory-keyword   = 'mandatory'
   max-elements-keyword = 'max-elements'
   min-elements-keyword = 'min-elements'
   module-keyword      = 'module'
   must-keyword        = 'must'
   namespace-keyword   = 'namespace'
   notification-keyword= 'notification'
   ordered-by-keyword  = 'ordered-by'
   organization-keyword= 'organization'
   output-keyword      = 'output'
   path-keyword        = 'path'
   pattern-keyword     = 'pattern'
   position-keyword    = 'position'
   prefix-keyword      = 'prefix'
   presence-keyword    = 'presence'
   range-keyword       = 'range'
   reference-keyword   = 'reference'
   refine-keyword      = 'refine'
```

```
   require-instance-keyword = 'require-instance'
   revision-keyword     = 'revision'
   revision-date-keyword = 'revision-date'
   rpc-keyword          = 'rpc'
   status-keyword       = 'status'
   submodule-keyword    = 'submodule'
   type-keyword         = 'type'
   typedef-keyword      = 'typedef'
   unique-keyword       = 'unique'
   units-keyword        = 'units'
   uses-keyword         = 'uses'
   value-keyword        = 'value'
   when-keyword         = 'when'
   yang-version-keyword= 'yang-version'
   yin-element-keyword = 'yin-element'

   ;; прочие ключевые слова
   add-keyword          = 'add'
   current-keyword      = 'current'
   delete-keyword       = 'delete'
   deprecated-keyword   = 'deprecated'
   false-keyword        = 'false'
   max-keyword          = 'max'
   min-keyword          = 'min'
   not-supported-keyword = 'not-supported'
   obsolete-keyword     = 'obsolete'
   replace-keyword      = 'replace'
   system-keyword       = 'system'
   true-keyword         = 'true'
   unbounded-keyword    = 'unbounded'
   user-keyword         = 'user'

   current-function-invocation = current-keyword *WSP "(" *WSP ")"

   ;; Базовые правила
   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

   ;; Идентификаторам НЕДОПУСТИМО начинаться с (('X'|'x') ('M'|'m') ('L'|'l'))
   identifier           = (ALPHA / "_")
                          *(ALPHA / DIGIT / "_" / "-" / ".")

   identifier-ref-arg-str = < a string that matches the rule
                            identifier-ref-arg >

   identifier-ref-arg   = [prefix ":"] identifier

   string               = < an unquoted string as returned by
                            the scanner >

   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              = ";" / "{" *unknown-statement "}"

   sep                  = 1*(WSP / line-break)
                          ; безусловный разделитель

   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
                          ; ' (одинарная кавычка)

   ;;
```

```
   ;; Основные правила RFC 5234
   ;;
   ALPHA               = %x41-5A / %x61-7A
                         ; A-Z / a-z

   CR                  = %x0D
                         ; возврат каретки

   CRLF                = CR LF
                         ; стандарт новой строки в Internet

   DIGIT               = %x30-39
                         ; 0-9

   DQUOTE              = %x22
                         ; " (двойная кавычка)

   HEXDIG              = DIGIT /
                         %x61 / %x62 / %x63 / %x64 / %x65 / %x66
                         ; только a..f в нижнем регистре

   HTAB                = %x09
                         ; горизонтальная табуляция

   LF                  = %x0A
                         ; перевод строки

   SP                  = %x20
                         ; пробел

   VCHAR               = %x21-7E
                         ; видимые (печатные) символы

   WSP                 = SP / HTAB
                         ; пустое пространство (пробельные символы)
   <CODE ENDS>
```

13. Сообщения об ошибках, связанных с YANG

Множество откликов NETCONF об ошибках определено для ошибок, связанных с обработкой модели данных. Если имеющий отношение к делу оператор YANG имеет субоператор error-app-tag, этот оператор заменяет принятое по умолчанию значение, описанное ниже.

13.1. Сообщение для данных, нарушающих unique

Если операция NETCONF будет давать конфигурацию, в которой ограничение unique перестаёт выполняться, возвращается описанная ниже ошибка.

```
   error-tag:      отказ операции
   error-app-tag:  данные не уникальны
   error-info:     <non-unique>: содержит экземпляр идентификатора,
                   указывающий на лист, нарушающий ограничение unique.
                   Этот элемент указывает 1 раз для каждого не
                   уникального leaf.

                   Элемент <non-unique> относится к пространству имён
                   YANG urn:ietf:params:xml:ns:yang:1.
```

13.2. Сообщение для данных, нарушающих max-elements

Если операция NETCONF будет создавать конфигурацию, где узел list или leaf-list включает слишком много элементов, возвращается описанная ниже ошибка.

```
   error-tag:      отказ операции
   error-app-tag:  слишком много элементов
```

Эта ошибка возвращается однократно и error-path указывает узел списка, даже при возникновении более одного избыточного потомка.

13.3. Сообщение для данных, нарушающих min-elements

Если операция NETCONF будет создавать конфигурацию, где узел list или leaf-list включает слишком мало элементов, возвращается описанная ниже ошибка.

```
   error-tag:      отказ операции
   error-app-tag:  слишком мало элементов
```

Эта ошибка возвращается однократно и error-path указывает узел списка, даже при возникновении более одного недостающего потомка.

13.4. Сообщение для данных, нарушающих must

Если операция NETCONF будет создавать конфигурацию, где нарушены ограничения, заданные оператором must, возвращается описанная ниже ошибка, если нет конкретного субоператора error-app-tag для оператора must.

```
   error-tag:      отказ операции
   error-app-tag:  нарушение must
```

13.5. Сообщение для данных, нарушающих require-instance

Если операция NETCONF будет создавать конфигурацию, где лист типа instance-identifier с оператором require-instance, имеющим значение true, указывает на несуществующий экземпляр, возвращается описанная ниже ошибка.

```
error-tag:      данные отсутствуют
error-app-tag:  требуется экземпляр
error-path:     путь к instance-identifier.
```

13.6. Сообщение для данных, не соответствующих типу leafref

Если операция NETCONF будет создавать конфигурацию, где лист типа leafref указывает на несуществующий экземпляр возвращается описанная ниже ошибка.

```
error-tag:      данные отсутствуют
error-app-tag:  требуется экземпляр
error-path:     путь к leafref.
```

13.7. Сообщение для данных, нарушающих обязательный choice

Если операция NETCONF будет создавать конфигурацию, где не существует узлов в обязательном выборе (choice), возвращается описанная ниже ошибка.

```
error-tag:      данные отсутствуют
error-app-tag:  отсутствует choice
error-path:     путь к элементу с отсутствующим choice.
error-info:     <missing-choice>: имя отсутствующего обязательного
                оператора choice.
                Элемент <missing-choice> относится к пространству имён
                YANG urn:ietf:params:xml:ns:yang:1.
```

13.8. Сообщение для данных, нарушающих insert

Если в <edit-config> используется insert и атрибут key или value для узла list или leaf-list и key или value указывает несуществующий экземпляр, возвращается описанная ниже ошибка.

```
error-tag:      некорректный атрибут
error-app-tag:  отсутствующий экземпляр
```

14. Взаимодействие с IANA

Этот документ определяет реестр для имён модуля и субмодулей YANG с именем YANG Module Names.

В этот реестр следует включать перечисленные ниже записи:

- имя модуля или субмодуля;
- назначенное для модуля пространство имён XML;
- префикс модуля;
- имя модуля, к которому относится субмодуль;
- ссылка на документацию (суб)модуля (например, номер RFC).

Реестр изначально пуст.

Для выделения значений требуется публикация RFC в соответствии с RFC 5226 [RFC5226]. Имена регистрируемых модулей YANG **должны** соответствовать правилам, приведённым в параграфе 6.2 и **должны** иметь префикс.

Префикс ietf- зарезервирован для документов IETF [RFC4844], irtf- зарезервирован для документов IRTF. Модули, публикуемые в других потоках RFC **должны** иметь аналогичный подходящий префикс.

Все имена регистрируемых модулей и субмодулей **должны** быть уникальными.

Все пространства имён XML в реестре **должны** быть уникальными.

Этот документ регистрирует два URI для пространств имён YANG YIN XML в реестре IETF XML [RFC3688]. В соответствии с форматом RFC 3688 добавлены приведённые ниже регистрации.

```
URI: urn:ietf:params:xml:ns:yang:yin:1
URI: urn:ietf:params:xml:ns:yang:1
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.
```

Этот документ регистрирует два новых типа, определенных в следующих параграфах.

14.1. Тип носителя application/yang

```
MIME media type name:  application
MIME subtype name:  yang
Mandatory parameters:  none
Optional parameters:  none
Encoding considerations:  8-bit
Security considerations:  See Section 15 in RFC 6020
Interoperability considerations:  None
Published specification:  RFC 6020
Applications that use this media type:
  YANG module validators, web servers used for downloading YANG
  modules, email clients, etc.
Additional information:
   Magic Number:  None
   File Extension:  .yang
```

```
    Macintosh file type code:  'TEXT'
  Personal and email address for further information:
    Martin Bjorklund <mbj@tail-f.com>
  Intended usage:  COMMON
  Author:
    This specification is a work item of the IETF NETMOD working group,
    with mailing list address <netmod@ietf.org>.
  Change controller:
    The IESG <iesg@ietf.org>
```

14.2. Тип носителя application/yin+xml

```
  MIME media type name:  application
  MIME subtype name:  yin+xml
  Mandatory parameters:  none
  Optional parameters:
    "charset":  This parameter has identical semantics to the charset
    parameter of the "application/xml" media type as specified in
    [RFC3023].
  Encoding considerations:
    Identical to those of "application/xml" as
    described in [RFC3023], Section 3.2.
  Security considerations:  See Section 15 in RFC 6020
  Interoperability considerations:  None
  Published specification:  RFC 6020
  Applications that use this media type:
    YANG module validators, web servers used for downloading YANG
    modules, email clients, etc.
  Additional information:
    Magic Number:  As specified for "application/xml" in [RFC3023],
                   Section 3.2.
    File Extension:  .yin
    Macintosh file type code:  'TEXT'
  Personal and email address for further information:
    Martin Bjorklund <mbj@tail-f.com>
  Intended usage:  COMMON
  Author:
    This specification is a work item of the IETF NETMOD working group,
    with mailing list address <netmod@ietf.org>.
  Change controller:
    The IESG <iesg@ietf.org>
```

15. Вопросы безопасности

Этот документ определяет язык для записи и чтения описаний управляющей информации. Язык сам по себе не оказывает влияния на безопасность Internet.

Применимы соображения по части безопасности, рассмотренные для протокола NETCONF (раздел 9 в [RFC6241]).

Данные, моделируемые в YANG, могут содержать деликатную информацию. RPC и уведомления, определённые в YANG могут служить для переноса деликатной информации.

Имеются вопросы безопасности, связанные с использованием данных, моделируемых в YANG. Такие вопросы следует рассматривать в документах, описывающих модели данных и документах для интерфейсом, применяемых для работы с данными (например, в документах NETCONF).

Данные, моделируемые в YANG, зависят от:

- безопасности передающей инфраструктуры, используемой для обмена деликатной информацией;
- безопасности приложений, которые хранят и выпускают такую деликатную информацию;
- адекватности механизмов аутентификации и контроля доступа для ограничения доступа к данным.

Анализаторы YANG должны быть отказоустойчивы к документам к некорректным форматированием. Чтение некорректно сформированных документов из неизвестных или недоверенных источников может приводить к получению атакующим привилегий пользователя, применяющего анализатор YANG. В предельном случае это может подвергать риску всю машину.

16. Участники работы

Ниже перечислены лица, внёсшие значимый вклад в разработку исходного документа YANG.

- Andy Bierman (Brocade)
- Balazs Lengyel (Ericsson)
- David Partain (Ericsson)
- Juergen Schoenwaelder (Jacobs University Bremen)
- Phil Shafer (Juniper Networks)

17. Благодарности

Редактор выражает благодарность всем, кто предоставил значимые комментарии к разным версиям этого документа: Mehmet Ersue, Washam Fan, Joel Halpern, Leif Johansson, Ladislav Lhotka, Gerhard Muenz, Tom Petch, Randy Presuhn, David Reid и Bert Wijnen.

18. Литература

18.1. Нормативные документы

[ISO.10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2003, 2003.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[XML-NAMES] Hollander, D., Tobin, R., Thompson, H., Bray, T., and A. Layman, "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] Malhotra, A. and P. Biron, "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. Дополнительная литература

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC3780] Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation Structure of Management Information", RFC 3780, May 2004.

[RFC4844] Daigle, L. and Internet Architecture Board, "The RFC Series and RFC Editor", RFC 4844, July 2007.

[XPATH2.0] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0", World Wide Web Consortium Recommendation REC-xpath20-20070123, January 2007, <http://www.w3.org/TR/2007/REC-xpath20-20070123>.

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

Адрес автора

Martin Bjorklund (редактор)
Tail-f Systems
EMail: mbj@tail-f.com

Перевод на русский язык

Николай Малых
nmalykh@protokols.ru