Internet Engineering Task Force (IETF)
Request for Comments: 7950
Category: Standards Track
ISSN: 2070-1721

M. Bjorklund, Ed.
Tail-f Systems

August 2016

Язык моделирования данных YANG 1.1

The YANG 1.1 Data Modeling Language

Аннотация

YANG представляет собой язык моделирования данных, применяемый в моделях конфигурационных данных, данных состояния, вызовов удалённых процедур (RPC[1]) и уведомлениях для протоколов управления сетями. Этот документ описывает синтаксис и семантику языка YANG версии 1.1. Эта версия устраняет неоднозначности и дефекты, обнаруженные в исходной спецификации языка YANG. Имеется незначительное число несовместимостей с языком YANG версии 1. Документ также задаёт отображения YANG на протокол управления конфигурацией сети NETCONF[2].

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[3] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[4]. Не все одобренные IESG документы претендуют на статус Internet Standard (раздел 2 в RFC 7841).

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке http://www.rfc-editor.org/info/rfc7950.

Авторские права

Авторские права (Copyright (c) 2016) принадлежат IETF Trust и лицам, указанным в качестве авторов документа. Все права защищены.

К этому документу применимы права и ограничения, перечисленные в BCP 78 и IETF Trust Legal Provisions и относящиеся к документам IETF (http://trustee.ietf.org/license-info), на момент публикации данного документа. Прочтите упомянутые документы внимательно, поскольку в них описаны права и ограничения, относящиеся к данному документу. Фрагменты программного кода, включённые в этот документ, распространяются в соответствии с упрощённой лицензией BSD, как указано в параграфе 4.e документа IETF Trust Legal Provisions, без каких-либо гарантий (как указано в Simplified BSD License).

Документ может содержать материалы из IETF Document или IETF Contribution, опубликованных или публично доступных до 10 ноября 2008 года. Лица, контролирующие авторские права на некоторые из таких документов, могли не предоставить IETF Trust права разрешать внесение изменений в такие документы за рамками процессов IETF Standards. Без получения соответствующего разрешения от лиц, контролирующих авторские права, этот документ не может быть изменён вне рамок процесса IETF Standards, не могут также создаваться производные документы за рамками процесса IETF Standards, за исключением форматирования документа для публикации или перевода с английского языка на другие языки.

Оглавление

1. Введение....6
1.1. Основные отличия от RFC 6020....6
2. Уровни требований....7
3. Терминология....7
3.1. О примерах....9
4. Обзор языка YANG....9
4.1. Функциональный обзор....9
4.2. Обзор языка....9
4.2.1. Модули и субмодули....9
4.2.2. Основы моделирования данных....10
4.2.2.1. Лист (leaf)....10
4.2.2.2. Лист-список (leaf-list)....10
4.2.2.3. Контейнер (container)....10
4.2.2.4. Список (list)....10
4.2.2.5. Пример модуля....11
4.2.3. Данные конфигурации и состояния....11
4.2.4. Встроенные типы....12
4.2.5. Производные типы (typedef)....12
4.2.6. Многократно используемые группы узлов (grouping)....12

[1]Remote Procedure Call.
[2]Network Configuration Protocol.
[3]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[4]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

4.2.7. Выбор (choice и case)....13
4.2.8. Расширение моделей данных (augment)....14
4.2.9. Определения операций....14
4.2.10. Определения уведомлений....15
5. Концепции языка....15
5.1. Модули и субмодули....15
5.1.1. Импорт и включение по номеру выпуска (revision)....16
5.1.2. Иерархии модулей....16
5.1.2.1. Представление NETCONF XML....16
5.2. Макет файла....17
5.3. Пространства имён XML....17
5.3.1. Пространство имён YANG XML....17
5.4. Преобразование имён группировок, типов и отождествлений....17
5.5. Вложенные определения типов и группировки....17
5.6. Соответствие....18
5.6.1. Базовое поведение....18
5.6.2. Необязательные функции....18
5.6.3. Отклонения от модели....18
5.6.4. Анонсирование информации о соответствии в NETCONF....18
5.6.5. Реализация модуля....19
5.7. Изменение хранилища данных....20
6. Синтаксис YANG....20
6.1. Лексическая маркировка....21
6.1.1. Комментарии....21
6.1.2. Маркеры....21
6.1.3. Кавычки....21
6.1.3.1. Примеры кавычек....21
6.2. Идентификаторы....22
6.2.1. Идентификаторы и пространства имён....22
6.3. Операторы....22
6.3.1. Расширения языка....22
6.4. Вычисление XPath....23
6.4.1. Контекст XPath....23
6.4.1.1. Примеры....24
6.5. Идентификатор узла схемы....25
7. Операторы YANG....25
7.1. Оператор module....25
7.1.1. Субоператоры для module....26
7.1.2. Оператор yang-version....26
7.1.3. Оператор namespace....26
7.1.4. Оператор prefix....26
7.1.5. Оператор import....26
7.1.5.1. Оператор revision-date для оператора import....27
7.1.6. Оператор include....27
7.1.7. Оператор organization....27
7.1.8. Оператор contact....27
7.1.9. Оператор revision....27
7.1.9.1. Субоператоры для revision....27
7.1.10. Пример использования....27
7.2. Оператор submodule....28
7.2.1. Субоператоры для submodule....28
7.2.2. Оператор belongs-to....29
7.2.3. Пример использования....29
7.3. Оператор typedef....29
7.3.1. Субоператоры для typedef....29
7.3.2. Оператор type....29
7.3.3. Оператор units....29
7.3.4. Оператор default....29
7.3.5. Пример использования....30
7.4. Оператор type....30
7.4.1. Субоператоры для type....30
7.5. Оператор container....30
7.5.1. Контейнеры....30
7.5.2. Субоператоры для container....30
7.5.3. Оператор must....31
7.5.4. Субоператоры для must....31
7.5.4.1. Оператор error-message....31
7.5.4.2. Оператор error-app-tag....31
7.5.4.3. Пример использования must и error-message....31
7.5.5. Оператор presence....32
7.5.6. Операторы дочерних узлов контейнера....32
7.5.7. Правила представления XML....32
7.5.8. Операции NETCONF <edit-config>....32
7.5.9. Пример использования....32
7.6. Оператор leaf....33
7.6.1. Значение листа по умолчанию....33
7.6.2. Субоператоры для leaf....33

7.6.3. Оператор type....33
7.6.4. Оператор default....33
7.6.5. Оператор mandatory....34
7.6.6. Правила представления XML....34
7.6.7. Операции NETCONF <edit-config>....34
7.6.8. Пример использования....34
7.7. Оператор leaf-list....34
7.7.1. Упорядочение....34
7.7.2. Принятые по умолчанию значения leaf-list....35
7.7.3. Субоператоры для leaf-list....35
7.7.4. Субоператор default для leaf-list....35
7.7.5. Оператор min-elements....35
7.7.6. Оператор max-elements....36
7.7.7. Оператор ordered-by....36
7.7.7.1. Системное упорядочение....36
7.7.7.2. Упорядочение пользователем....36
7.7.8. Правила представления XML....36
7.7.9. Операции NETCONF <edit-config>....36
7.7.10. Пример использования....37
7.8. Оператор list....37
7.8.1. Субоператоры для list....37
7.8.2. Оператор key....38
7.8.3. Оператор unique....38
7.8.3.1. Пример использования....38
7.8.4. Операторы дочерних узлов списка....39
7.8.5. Правила представления XML....39
7.8.6. Операции NETCONF <edit-config>....39
7.8.7. Пример использования....39
7.9. Оператор choice....41
7.9.1. Субоператоры для choice....41
7.9.2. Оператор case для выбора....41
7.9.2.1. Субоператоры для case....42
7.9.3. Субоператор default для выбора....42
7.9.4. Оператор mandatory для выбора....43
7.9.5. Правила представления XML....43
7.9.6. Пример использования....43
7.10. Оператор anydata....43
7.10.1. Субоператоры для anydata....44
7.10.2. Правила представления XML....44
7.10.3. Операции NETCONF <edit-config>....44
7.10.4. Пример использования....44
7.11. Оператор anyxml....44
7.11.1. Субоператоры для anyxml....45
7.11.2. Правила представления XML....45
7.11.3. Операции NETCONF <edit-config>....45
7.11.4. Пример использования....45
7.12. Оператор grouping....45
7.12.1. Субоператоры для grouping....46
7.12.2. Пример использования....46
7.13. Оператор uses....46
7.13.1. Субоператоры для uses....46
7.13.2. Оператор refine....46
7.13.3. Правила представления XML....47
7.13.4. Пример использования....47
7.14. Оператор rpc....47
7.14.1. Субоператоры для rpc....47
7.14.2. Оператор input....48
7.14.2.1. Субоператоры для input....48
7.14.3. Оператор output....48
7.14.3.1. Субоператоры для output....48
7.14.4. Правила представления NETCONF XML....48
7.14.5. Пример использования....49
7.15. Оператор action....49
7.15.1. Субоператоры для action....49
7.15.2. Правила представления NETCONF XML....49
7.15.3. Пример использования....50
7.16. Оператор notification....50
7.16.1. Субоператоры для notification....51
7.16.2. Правила представления NETCONF XML....51
7.16.3. Пример использования....51
7.17. Оператор augment....52
7.17.1. Субоператоры для augment....52
7.17.2. Правила представления XML....53
7.17.3. Пример использования....53
7.18. Оператор identity....54
7.18.1. Субоператоры для identity....54
7.18.2. Оператор base....54

7.18.3. Пример использования....54
7.19. Оператор extension....55
7.19.1. Субоператоры для extension....55
7.19.2. Оператор argument....55
7.19.2.1. Субоператор для argument....55
7.19.2.2. Оператор yin-element....55
7.19.3. Пример использования....55
7.20. Операторы, связанные с соответствием спецификации....55
7.20.1. Оператор feature....55
7.20.1.1. Субоператоры для feature....56
7.20.2. Оператор if-feature....56
7.20.2.1. Пример использования....56
7.20.3. Оператор deviation....56
7.20.3.1. Субоператоры для deviation....57
7.20.3.2. Оператор deviate....57
7.20.3.3. Пример использования....57
7.21. Субоператоры общего назначения....58
7.21.1. Оператор config....58
7.21.2. Оператор status....58
7.21.3. Оператор description....58
7.21.4. Оператор reference....58
7.21.5. Оператор when....59
8. Ограничения....59
8.1. Ограничения для данных....59
8.2. Изменение данных конфигурации....60
8.3. Модель применения ограничений NETCONF....60
8.3.1. Анализ данных....60
8.3.2. Обработка NETCONF <edit-config>....60
8.3.3. Проверка пригодности....60
9. Встроенные типы....60
9.1. Каноническое представление....61
9.2. Целочисленные встроенные типы....61
9.2.1. Лексическое представление....61
9.2.2. Каноническая форма....61
9.2.3. Ограничения....61
9.2.4. Оператор range....61
9.2.4.1. Субоператоры для range....62
9.2.5. Пример использования....62
9.3. Встроенный тип decimal64....62
9.3.1. Лексическое представление....62
9.3.2. Каноническая форма....62
9.3.3. Ограничения....62
9.3.4. Оператор fraction-digits....62
9.3.5. Пример использования....63
9.4. Встроенный тип string....63
9.4.1. Лексическое представление....63
9.4.2. Каноническая форма....63
9.4.3. Ограничения....63
9.4.4. Оператор length....63
9.4.4.1. Субоператоры для length....63
9.4.5. Оператор pattern....63
9.4.5.1. Субоператоры для pattern....63
9.4.6. Оператор modifier....63
9.4.7. Пример использования....63
9.5. Встроенный тип boolean....64
9.5.1. Лексическое представление....64
9.5.2. Каноническая форма....64
9.5.3. Ограничения....64
9.6. Встроенный тип enumeration....64
9.6.1. Лексическое представление....64
9.6.2. Каноническая форма....64
9.6.3. Ограничения....64
9.6.4. Оператор enum....64
9.6.4.1. Субоператоры для enum....65
9.6.4.2. Оператор value....65
9.6.5. Пример использования....65
9.7. Встроенный тип bits....66
9.7.1. Ограничения....66
9.7.2. Лексическое представление....66
9.7.3. Каноническая форма....66
9.7.4. Оператор bit....66
9.7.4.1. Субоператоры для bit....66
9.7.4.2. Оператор position....66
9.7.5. Пример использования....66
9.8. Встроенный тип binary....67
9.8.1. Ограничения....67
9.8.2. Лексическое представление....67

9.8.3. Каноническая форма....67
9.9. Встроенный тип leafref....67
9.9.1. Ограничения....67
9.9.2. Оператор path....67
9.9.3. Оператор require-instance....68
9.9.4. Лексическое представление....68
9.9.5. Каноническая форма....68
9.9.6. Пример использования....68
9.10. Встроенный тип identityref....69
9.10.1. Ограничения....69
9.10.2. Оператор base для identityref....69
9.10.3. Лексическое представление....70
9.10.4. Каноническая форма....70
9.10.5. Пример использования....70
9.11. Встроенный тип empty....70
9.11.1. Ограничения....70
9.11.2. Лексическое представление....70
9.11.3. Каноническая форма....71
9.11.4. Пример использования....71
9.12. Встроенный тип union....71
9.12.1. Ограничения....71
9.12.2. Лексическое представление....71
9.12.3. Каноническая форма....71
9.12.4. Пример использования....71
9.13. Встроенный тип instance-identifier....72
9.13.1. Ограничения....72
9.13.2. Лексическое представление....72
9.13.3. Каноническая форма....72
9.13.4. Пример использования....72
10. Функции XPath....72
10.1. Функция для набора узлов....72
10.1.1. current()....72
10.1.1.1. Пример использования....72
10.2. Функция для строк....73
10.2.1. re-match()....73
10.2.1.1. Пример использования....73
10.3. Функция для типов leafref и instance-identifier....73
10.3.1. deref()....73
10.3.1.1. Пример использования....73
10.4. Функции для типа identityref....74
10.4.1. derived-from()....74
10.4.1.1. Пример использования....74
10.4.2. derived-from-or-self()....74
10.4.2.1. Пример использования....74
10.5. Функции для типа enumeration....74
10.5.1. enum-value()....74
10.5.1.1. Пример использования....74
10.6. Функции для типа bits....75
10.6.1. bit-is-set()....75
10.6.1.1. Пример использования....75
11. Обновление модулей....75
12. Сосуществование с YANG версии 1....76
13. YIN....76
13.1. Формальное определение YIN....77
13.1.1. Пример использования....78
14. Грамматика ABNF для YANG....78
15. Отклики NETCONF об ошибках, связанных с YANG....91
15.1. Сообщение для данных, нарушающих unique....92
15.2. Сообщение для данных, нарушающих max-elements....92
15.3. Сообщение для данных, нарушающих min-elements....92
15.4. Сообщение для данных, нарушающих must....92
15.5. Сообщение для данных, нарушающих require-instance....92
15.6. Сообщение для данных, нарушающих обязательный choice....92
15.7. Сообщение для данных, нарушающих insert....92
16. Взаимодействие с IANA....92
17. Вопросы безопасности....92
18. Литература....93
18.1. Нормативные документы....93
18.2. Дополнительная литература....93
Благодарности....94
Участники работы....94
Адрес автора....94

1. Введение

YANG представляет собой язык моделирования данных, изначально предназначенный для работы с конфигурационными параметрами и данными состояния протокола настройки конфигурации сети NETCONF, вызовов удалённых процедур NETCONF RPC и уведомлений NETCONF [RFC6241]. С момента публикации YANG версии 1 [RFC6020] язык применялся или был предложен для использования также в других протоколах (например, RESTCONF [RESTCONF] и CoMI[1] [CoMI]). Кроме того, было предложено кодирование, отличающееся от XML (например, JSON [RFC7951]).

В этом документе описывается синтаксис и семантика языка YANG версии 1.1. Описано также представление модулей YANG на расширяемом языке разметки XML[2] [XML] и использование операций NETCONF для манипулирования данными. Возможно использование и других протоколов, но это выходит за рамки спецификации.

В документе [YANG-Guidelines] приведены некоторые рекомендации по разработке моделей данных YANG.

Следует подчеркнуть, что этот документ не отменяет RFC 6020 [RFC6020].

1.1. Основные отличия от RFC 6020

Этот документ определяет версию 1.1 языка YANG, которая исправляет неточности и дефекты исходной спецификации языка YANG [RFC6020].

Ниже перечислены изменения, не совместимые с YANG версии 1.

- Изменены правила интерпретации escape-символов внутри двойных кавычек. Это изменение не совместимо с YANG версии 1 и при переходе к версии 1.1 модули, использующие непригодные в новой версии последовательности символов, должны быть изменены в соответствии с новыми правилами. Подробности приведены в параграфе 6.1.3. Кавычки.
- Строки, не заключённые в кавычки, не могут включать символы одинарных или двойных кавычек. Это изменение не совместимо с YANG версии 1 и при переходе к версии 1.1 модули, использующие непригодные в новой версии последовательности символов, должны быть изменены в соответствии с новыми правилами. Подробности приведены в параграфе 6.1.3. Кавычки.
- Конструкции when и if-feature не применимы в списке ключей. Это изменение не совместимо с YANG версии 1 и при переходе к версии 1.1 модули, использующие непригодные в новой версии конструкции, должны быть удалены для соответствия новым правилам.
- Определены допустимые в модулях YANG символы. При переходе к версии 1.1 все символы, ставшие непригодными, должны быть удалены. Подробности приведены в разделе 6. Синтаксис YANG.
- Запрещено использование несимвольных кодов (noncharacter) во встроенном типе string. Это изменение оказывает влияние на работу протоколов на базе YANG.

Ниже перечислены другие изменения, внесённые в язык YANG.

- Номер версии YANG 1 заменён на 1.1.
- Оператор yang-version стал обязательным в версии YANG 1.1.
- Расширен синтаксис if-feature для использования логических выражений с именами функций.
- Разрешено использовать if-feature в bit, enum и identity.
- Разрешено использовать if-feature в refine.
- Разрешено использовать choice в качестве сокращённого оператора case (параграф 7.9.2).
- Добавлен субоператор modifier в оператор pattern (9.4.6. Оператор modifier).
- Разрешено использовать must в input, output и notification.
- Разрешено использовать require-instance в leafref.
- Разрешено использовать description и reference в import и include.
- Разрешён импорт множества выпусков модуля.
- Разрешено использовать augment для добавления условно обязательных узлов (7.17. Оператор augment).
- Добавлен набор функций XPath[3] в разделе 10. Функции XPath.
- Уточнён XPath дерева контекста в параграфе 6.4.1. Контекст XPath.
- Определено строковое значение identityref в выражениях XPath (9.10. Встроенный тип identityref).
- Уточнено значение неожиданных имён в leafrefs определений типов (typedefs) (параграфы 6.4.1 и 9.9.2).
- Разрешено создание идентификационных данных (identity) из множества базовых отождествлений (параграфы 7.18. Оператор identity и 9.10. Встроенный тип identityref).
- Разрешено применять перечисляемые (enumeration) и биты (bit) в качестве субтипов (параграфы 9.6 и 9.7).
- Разрешены принятые по умолчанию значения leaf-list (7.7.2. Принятые по умолчанию значения leaf-list).
- Разрешены совпадающие (не уникальные) значения в leaf-list, не относящихся к конфигурации (параграф 7.7).

[1] Constrained Application Protocol (CoAP) Management Interface - интерфейс управления протоколом ограниченных приложений.
[2] Extensible Markup Language.
[3] XML Path Language.

- В грамматике применяется синтаксис регистро-чувствительных строк (как в [RFC7405]).
- Изменён механизм анонсирования модулей (5.6.4. Анонсирование информации о соответствии в NETCONF).
- Изменены правила области действия для определений в субмодулях. Субмодуль сейчас может указывать на все определения во всех субмодулях, относящихся к тому же модулю, без применения оператора include.
- Добавлен оператор action, который используется для определения операций, привязанных к узлам данных.
- Разрешено привязывать уведомления к узлам данных.
- Добавлен оператор определения данных anydata (7.10. Оператор anydata), который **рекомендуется** применять взамен anyxml, когда данные могут моделироваться в YANG.
- Разрешены типы empty и leafref в объединениях (union).
- Разрешён тип empty в ключах (key).
- Удалено ограничение, не позволявшее идентификаторам начинаться с символов «xml».

Ниже приведено изменение, внесённое в отображение для протокола NETCONF.

- Сервер анонсирует поддержку модулей YANG 1.1, используя ietf-yang-library [RFC7895] вместо её перечисления в списке возможностей сообщения <hello>.

2. Уровни требований

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

3. Терминология

Ниже приведены определения используемых в документе терминов.

action - действие
Операция, определённая для узла в дереве данных.

anydata - любые данные
Узел данных, который может содержать неизвестный набор узлов, которые могут моделироваться языком YANG, за исключением anyxml.

anyxml - любые данные XML
Узел данных, который может содержать любой неизвестный блок (chunk) данных XML.

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

base type - базовый тип
Тип, на основе которого создаются другие типы. Может быть встроенным или производным от другого типа.

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

choice - выбор
Узел схемы, где действует лишь один из указанных вариантов.

client - клиент
Объект, который может иметь доступ к определенным YANG данным на сервере с использованием того или иного сетевого протокола.

conformance - соответствие
Степень точности соблюдения сервером модели данных.

container - контейнер
Внутренний узел данных, существующий в дереве не более чем в одном экземпляре. Контейнер не имеет значения, но имеет набор дочерних узлов.

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

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

data node - узел данных
Узел в дереве схемы, который может быть помещён в дерево данных, один из container, leaf, leaf-list, list, anydata и anyxml.

data tree - дерево данных
Воплощённое дерево любых данных, промоделированных с помощью YANG, например, конфигурационные параметры, данные состояния, комбинация данных конфигурации и состояния, входные данные операции или RPC, выходные данные операции или RPC, уведомление.

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

extension - расширение
Расширение добавляет к операторам не включённую в YANG семантику. Оператор extension определяет новые операторы для выражения этой семантики.

feature - функция, возможность
Механизм для маркировки части модели, как необязательной. Определения могут помечаться именем функции и применимы только для серверов, которые эту функцию поддерживают.

grouping - группировка
Набор многократно используемых (reusable) узлов схемы, который может применяться 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.

identity - отождествление
Уникальное в глобальном масштабе, абстрактное и не типизованное имя.

instance identifier - идентификатор экземпляра
Механизм для идентификации конкретных узлов в дереве данных.

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

leaf - лист
Узел данных, который существует в дереве данных не более чем в одном экземпляре. 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.

mandatory node - обязательный узел
Один из перечисленных ниже вариантов:
- узел leaf, choice, anydata или anyxml с оператором mandatory, имеющим значение true;
- узел list или leaf-list с оператором min-elements, имеющим значение больше 0;
- узел container без оператора presence, имеющий в качестве потомка хотя бы один обязательный узел.

module - модуль
Модуль YANG определяет иерархию узлов схемы. Со своими и импортированными или включёнными из любого источника определениями модуль является самодостаточным и «компилируемым».

non-presence container - контейнер без присутствия
Контейнер, который не имеет смысла сам по себе и служит только для включения дочерних узлов.

presence container - контейнер присутствия
Контейнер, само присутствие которого имеет некий смысл (значение).

RPC
Вызов удалённой процедуры.

RPC operation - операция RPC
Вызов конкретной удалённой процедуры (RPC).

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

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

schema tree - дерево схемы
Иерархия определения, заданная внутри модуля.

server - сервер
Объект, обеспечивающий клиентам доступ к определенным в YANG данным через тот или иной протокол сетевого управления.

server deviation - отклонение (в поведении) сервера
Отказ сервера реализовать модуль.

submodule - субмодуль
Частичное определение модуля, вносящее производные типы, группировки, узлы данных, RPC, действия и уведомления для модуля. Модуль YANG может включать 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, служащий для создания набора узлов схемы, определённого в операторе grouping. Создаваемые узлы могут быть уточнены и дополнены в соответствии с конкретными потребностями.

value space - пространство значений
Применительно к типу данных - набор значений, разрешённых для этого типа. Для экземпляра leaf или leaf-list - пространство значений его типа данных.

Перечисленные ниже термины определены в [RFC6241]:

- конфигурационные данные (configuration data);
- конфигурационное хранилище (configuration datastore);
- хранилище данных (datastore);
- данные состояния (state data).

При моделировании в YANG хранилище данных реализуется в виде экземпляра дерева данных, конфигурационное хранилище данных реализуется в виде экземпляра дерева данных с параметрами конфигурации.

3.1. О примерах

В документе встречаются многочисленные примеры операторов YANG. Предполагается, что эти примеры будут иллюстрировать отдельные функции, но не являются законченными модулями YANG.

4. Обзор языка YANG

Этот раздел, не являющийся нормативным, содержит ознакомительный обзор языка YANG.

4.1. Функциональный обзор

Язык YANG был разработан для моделирования данных в протоколе NETCONF. Модуль YANG определяет иерархию 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. Хотя это и выходит за рамки спецификации, YANG можно использовать и с другими протоколами.

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 своих субмодулей. Иерархия может дополняться путём разрешения 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, задавая присутствие или значения узлов в зависимости от наличия или значения других узлов в иерархии. Эти ограничения применяются клиентом или сервером.

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 таких механизмов, как ограничения на число 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). Такие группировки позволяют уточнять или дополнять узлы, обеспечивая возможность адаптации к конкретным потребностям. Производные типы и группировки могут определяться в одном модуле и применяться в нем же или в импортировавшем его другом модуле.

Конструкции иерархии данных YANG включают определения списков, которые указываются 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] (13. YIN), 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.

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 представлением моделей данных для протоколов управления сетями такими как NETCONF, а не произвольных документов XML или моделей данных.

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) [RFC6643]. Однако YANG не обеспечивает трансляции в SMIv2.

4.2. Обзор языка

В этом разделе вводятся некоторые важные конструкции YANG, которые будут нужны для понимания специфики языка в последующих параграфах.

4.2.1. Модули и субмодули

Модели данных YANG определяются в модулях. Модуль содержит набор связанных между собой определений.

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

Сервер 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 задачами разработчика. Извне это будет по-прежнему выглядеть единым модулем, независимо от наличия и размера субмодулей.

Оператор import позволяет модулю или субмодулю указывать определения из других модулей.

Оператор include используется в модуле для указания входящих в него субмодулей.

[1] YANG Independent Notation - независимая нотация YANG.
[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. Основы моделирования данных

YANG определяет четыре основных типа узлов данных для моделирования. В каждом из последующих параграфов примеры показывают синтаксис YANG и соответствующее представление XML. Синтаксис операторов YANG определён в параграфе 6.3. Операторы.

4.2.2.1. Лист (leaf)

Экземпляр листа содержит простые данные, такие как целое число или строка. Лист имеет в точности одно значение конкретного типа и не имеет дочерних узлов.

Пример YANG.

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

Пример представления XML.

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

Оператор leaf описан в параграфе 7.6. Оператор leaf.

4.2.2.2. Лист-список (leaf-list)

Лист-список определяет последовательность значений определённого типа.

Пример YANG.

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

Пример представления 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. Оператор leaf-list.

4.2.2.3. Контейнер (container)

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, действия и уведомления).

Пример YANG.

```
container system {
  container login {
    leaf message {
      type string;
      description
        "Сообщение, выдаваемое при старте сессии регистрации.";
    }
  }
}
```

Пример представления XML.

```
<system>
  <login>
    <message>Добрый день!</message>
  </login>
</system>
```

Оператор container описан в разделе параграфе 7.5. Оператор container.

4.2.2.4. Список (list)

Список определяет последовательность элементов. Каждый элемент подобен контейнеру и однозначно идентифицируется значениями ключевых листьев, если они определены. 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 представления 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. Оператор list.

4.2.2.5. Пример модуля

Ниже показан пример комбинации операторов для определения модуля.

```
// Содержимое примера example-system.yang
module example-system {
  yang-version 1.1;
  namespace "urn:example:system";
  prefix "sys";

  organization "Example Inc.";
  contact "joe@example.com";
  description
    "Модуль для сущностей, реализующих систему Example.";

  revision 2007-06-09 {
    description "Исходный 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 сессии регистрации.";
      }

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

4.2.3. Данные конфигурации и состояния

YANG может моделировать данные состояния или конфигурации в зависимости от оператора config. Когда узел имеет тег config false, это означает, что его субиерархия помечена как данные состояния. Если узел имеет метку config true, его субиерархия содержит данные конфигурации. Родительские контейнеры, списки и ключевые узлы также указываются как содержащие контекст для данных состояния.

В приведённом ниже примере для каждого интерфейса определены два узла, указывающие заданную в конфигурации и наблюдаемую скорость.

```
list interface {
  key "name";
  config true;

  leaf name {
    type string;
```

```
      }
      leaf speed {
        type enumeration {
          enum 10m;
          enum 100m;
          enum auto;
        }
      }
      leaf observed-speed {
        type uint32;
        config false;
      }
    }
```

Оператор config описан в параграфе 7.21.1. Оператор config.

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. Встроенные типы.

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. Оператор type.

4.2.5. Производные типы (typedef)

YANG позволяет определять производные 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";
  }
}

leaf completed {
  type percent;
}
```

Пример представления XML

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

Оператор typedef описан в параграфе 7.3. Оператор typedef.

4.2.6. Многократно используемые группы узлов (grouping)

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

Пример YANG

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

container peer {
  container destination {
```

```
        uses target;
      }
    }
```

Пример представления XML

```
    <peer>
      <destination>
        <address>2001:db8::2</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 7.12. Оператор grouping.

4.2.7. Выбор (choice и case)

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.

Узлы choice и case появляются только в дереве схемы и не могут присутствовать в дереве данных. Дополнительные уровни иерархии за пределами концептуальной схемы не требуются. Присутствие варианта указывается наличием одного или множества узлов внутри оператора.

Поскольку в каждый момент может быть приемлем лишь один из вариантов (case) выбора (choice), при создании в дереве данных узла из одного варианта (case), узлы из всех остальных вариантов неявно удаляются. Сервер обеспечивает выполнение этого ограничения, предотвращая возникновение несовместимости в конфигурации.

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 представления XML

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

Оператор choice описан в параграфе 7.9. Оператор choice.

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.

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

Когда сервер реализует модуль с оператором augment, это предполагает, что реализация сервером дополненного модуля включает добавленные узлы.

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, элементы XML, добавляемые в представление, используют пространство имён дополняющего модуля. Например, если приведённый выше пример дополнения происходил в модуле с префиксом other, XML будет выглядеть, как показано ниже.

```
<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 7.17. Оператор augment.

4.2.9. Определения операций

Язык YANG позволяет определять операции. Имена операций, а также входные и выходные параметры моделируются с использованием операторов определения данных YANG. Операции на верхнем уровне модуля определяются с помощью оператора rpc. Операции могут также привязываться к узлам данных типа container и list. Такие операции определяются с помощью оператора action.

Ниже приведён пример определения операции YANG верхнего уровня.

```
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://example.com/system">
    <image-name>example-fw-2.3</image-name>
  </activate-software-image>
</rpc>

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

Ниже представлен пример определения операции, привязанной к узлу данных типа list.

```
list interface {
  key "name";

  leaf name {
    type string;
  }

  action ping {
    input {
      leaf destination {
        type inet:ip-address;
      }
    }
    output {
```

```
          leaf packet-loss {
            type uint8;
          }
        }
      }
    }
```

Пример NETCONF XML

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <interface xmlns="http://example.com/system">
      <name>eth1</name>
      <ping>
        <destination>192.0.2.1</destination>
      </ping>
    </interface>
  </action>
</rpc>

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

Оператор rpc описан в параграфе 7.14. Оператор rpc, action - в параграфе 7.15. Оператор action.

4.2.10. Определения уведомлений

Язык YANG позволяет определять уведомления. Для задания содержимого уведомления применяются операторы определения данных YANG.

Пример YANG

```
notification link-failure {
  description
    "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="urn:example:system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

Оператор notification описан в параграфе 7.16. Оператор notification.

5. Концепции языка

5.1. Модули и субмодули

Модуль является базовым блоком определений в YANG и определяет одну модель данных. Модуль может 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.

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. В рамках сервера все модули **должны** иметь уникальные имена.

Модули используют оператор include для перечисления всех своих субмодулей. Модуль или относящиеся к нему субмодули могут ссылаться на определения этого модуля и всех включённых в него субмодулей.

Модули и субмодули используют оператор import для ссылок на внешние модули. Операторы в модуле и субмодулях могут указывать определения из внешних модулей, используя префикс, указанный в операторе import.

Для совместимости с YANG версии 1 субмодуль **может** использовать оператор include для ссылки на другие субмодули того же модуля, но это не требуется в YANG версии 1.1. Субмодуль может ссылаться на любое определение в содержащем его модуле и всех его субмодулях. В субмодули **недопустимо** включать выпуски (revision) других субмодулей, отличающиеся от включённых в модуль.

В модули и субмодули **недопустимо** включать субмодули из других модулей, а субмодулям недопустимо импортировать включающий их (свой) модуль.

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

- Для того, чтобы модуль или субмодуль мог ссылаться на определения из внешнего модуля, этот внешний модуль **должен** быть импортирован (import).
- Модуль **должен** включать (include) все свои субмодули.
- Модуль и его субмодули **могут** ссылаться на определения этого модуля и всех включённых в него субмодулей.

Недопустимо создавать замкнутые цепочки импорта. Например, если модуль a импортирует модуль b, то b не может импортировать 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 эти типы данных (например, int32) префиксы применяться не могут. Синтаксис ссылок на определения формально задан правилом identifier-ref в разделе 14. Грамматика ABNF для YANG.

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). Изначально в модуль импортируются 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, может 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 {
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

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

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

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

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

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

Если модуль импортируется без указания выпуска, используемый им выпуск не определён.

5.1.2. Иерархии модулей

YANG позволяет моделировать данные с множеством иерархий, когда данные могут иметь более одного узла верхнего уровня. Каждый узел верхнего уровня в модели определяет отдельную иерархию. Модели с множеством узлов верхнего уровня иногда очень удобны и поддерживаются YANG.

5.1.2.1. Представление NETCONF XML

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 example-config {
  yang-version 1.1;
  namespace "urn:example: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="urn:example:config">
        <!-- системные данные -->
      </system>
      <routing xmlns="urn:example:config">
        <!-- данные маршрутизации -->
      </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' )
```

Элемент module-or-submodule-name указывает имя модуля или субмодуля, необязательный элемент revision-date - последний выпуск модуля или субмодуля, заданный оператором revision (параграф 7.1.9).

Расширение имени .yang говорит о том, что содержимое этого файла использует синтаксис YANG (6. Синтаксис YANG), а .yin указывает использование синтаксиса YIN (13. YIN).

Анализаторы YANG могут находить импортируемые модули и включаемые субмодули на базе этих соглашений.

5.3. Пространства имён XML

Все определения YANG задаются внутри модуля. Каждый модуль привязан 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, опубликованных в серии RFC [RFC4844], **должны** выделяться IANA (см раздел 14 в [RFC6020]).

Пространства имён XML для приватных модулей должны выделяться 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. Оператор namespace.

5.3.1. Пространство имён YANG XML

YANG определяет пространство имён XML для операций NETCONF <edit-config>, содержимого <error-info> и элемента <action>. 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 удовлетворения всех требований, заданных исходным определением. Это похоже на область действия (scope) в традиционных языках программирования.

Например, если модуль имеет группировку, в которой присутствует ссылка на тип, при использовании этой группировки в другом модуле тип преобразуется (resolve) в контексте исходного (а не второго) модуля. За счёт этого не возникает неоднозначности 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.

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. указанные 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 путём нахождения соответствующего оператора typedef или grouping среди непосредственных субоператоров каждого оператора-предка.

5.6. Соответствие

Соответствие модели показывает, насколько точно сервер следует этой модели. Вообще говоря, сервер отвечает за правильную реализацию модели и это позволяет приложениям считать, что серверы одинаково реализуют модель. Отклонения от модели могут сузить возможности и увеличить число проблем для использующих модель 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 на базе YANG - это соглашение 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. Необязательные функции

Во многих моделях авторы делают некоторые части модели необязательными (условными). Сервер 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 сервера хранение системных журналов.

YANG поддерживает механизм условной поддержки с помощью конструкции feature. Это даёт разработчику модели механизм, позволяющий сделать отдельные части модели условными с решением вопроса их поддержки на сервере. Модель может содержать конструкции, которые поддерживаются не всеми серверами. 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 делать соответствующие части модуля необязательными. Если сервер поддерживает функцию, соответствующая часть модуля становится действительной для этого сервера. Если же функция не поддерживается, эта часть модуля становится непригодной для применения и приложения учитывают это.

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

Дополнительная информация о необязательных функциях приведена в параграфе 7.20.1. Оператор feature.

5.6.3. Отклонения от модели

В идеальном мире от всех серверов требуется точная реализация модели и отклонения от неё не разрешаются. Однако в реальности серверы зачастую просто не способны или не предназначены для полной реализации модели. В автоматизации на базе YANG требуются механизмы, с помощью которых серверы 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 заранее знать об этом и принять соответствующие меры.

Отклонения серверов 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 7.20.3. Оператор deviation.

5.6.4. Анонсирование информации о соответствии в NETCONF

В этом параграфе описан механизм анонсирования сведений о совместимости. В будущих спецификациях могут быть определены другие механизмы.

Сервер NETCONF **должен** анонсировать реализуемые им модули (5.6.5. Реализация модуля) путём реализации модуля YANG ietf-yang-library, определённого в [RFC7895], и перечисления всех реализованных модулей в списке /modules-state/module.

Сервер также **должен** анонсировать в сообщении <hello> приведённую ниже возможность.

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

Параметр revision имеет такое же значение, как дата выпуска модуля ietf-yang-library, реализованного сервером, и **должен** присутствовать.

Параметр module-set-id имеет такое же значение, как лист /modules-state/module-set-id из модуля ietf-yang-library, и **должен** присутствовать.

С помощью этого механизма клиент может кэшировать поддерживаемые сервером модули и обновлять кэш лишь при изменении значения module-set-id в сообщении <hello>.

5.6.5. Реализация модуля

Сервер поддерживает модуль, если в нем реализованы узлы данных, RPC, действия (action), уведомления и отклонения (deviation) для этого модуля.

Серверам **недопустимо** реализовать одновременно более одного выпуска модуля.

Если сервер реализует модуль A, который импортирует модуль B, и A применяет любой из узлов модуля B в операторе augment или path, который поддерживает сервер, этот сервер **должен** реализовать выпуск модуля B, в котором эти узлы определены. Это не зависит от того, импортирован ли модуль B по выпуску.

Если сервер реализует модуль A, который импортирует модуль C без указания даты выпуска модуля C, и сервер не реализует C (например, если C содержит лишь операторы typedef), он **должен** указать модуль C в списке /modules-state/module из ietf-yang-library [RFC7895] и **должен** установить для этого модуля conformance-type import.

Если сервер указывает модуль C в списке /modules-state/module из ietf-yang-library и имеются другие модули M, которые импортируют C без указания выпуска модуля C, сервер **должен** использовать определения из последней версии модуля C, указанной для модулей M.

Эти правила обусловлены тем, что клиенты должны иметь возможность узнать конкретную структуру и типы всех leaf и leaf-list модели данных, реализованных на сервере.

Рассмотрим в качестве примера приведённые ниже модули.

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

  import b {
    revision-date 2015-01-01;
  }
  import c;

  revision 2015-01-01;

  feature foo;

  augment "/b:x" {
    if-feature foo;

    leaf y {
      type b:myenum;
    }
  }

  container a {
    leaf x {
      type c:bar;
    }
  }
}

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

  revision 2015-01-01;

  typedef myenum {
    type enumeration {
      enum zero;
    }
  }

  container x {
  }
}

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

  revision 2015-04-04;
  revision 2015-01-01;

  typedef myenum {
    type enumeration {
      enum zero; // добавлено 2015-01-01
```

```
          enum one;  // добавлено 2015-04-04
        }
      }

      container x {  // добавлено 2015-01-01
        container y; // добавлено 2015-04-04
      }
    }

    module c {
      yang-version 1.1;
      namespace "urn:example:c";
      prefix "c";

      revision 2015-02-02;

      typedef bar {
        ...
      }
    }

    module c {
      yang-version 1.1;
      namespace "urn:example:c";
      prefix "c";

      revision 2015-03-03;
      revision 2015-02-02;

      typedef bar {
        ...
      }
    }
```

Сервер, который реализует выпуск 2015-01-01 модуля a и поддерживает функцию foo, может реализовать выпуск 2015-01-01 или 2015-04-04 для модуля b. Поскольку модуль b был импортирован выпуском, тип листа /b:x/a:y не меняется в зависимости от используемого сервером выпуска модуля b.

Сервер, который реализует модуль a, но не поддерживает функцию foo, не будет реализовать модуль b.

Сервер, реализующий выпуск 2015-01-01 модуля a, указывает любой выпуск модуля c и перечисляет его в списке /modules-state/module из ietf-yang-library.

Приведённый ниже пример кода XML показывает пригодные данные для списка /modules-state/module сервера, который реализует модуль a.

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

  <module>
    <name>c</name>
    <revision>2015-02-02</revision>
    <namespace>urn:example:c</namespace>
    <conformance-type>import</conformance-type>
  </module>
</modules-state>
```

5.7. Изменение хранилища данных

Модели данных могут разрешать серверу менять хранилище конфигурации способами, которые не заданы явно в сообщениях протокола сетевого управления. 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 и 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.

Допустимыми символами для модулей YANG являются символы Unicode и ISO/IEC 10646 [ISO.10646], включая символы табуляции, возврата каретки и перевода строки, но исключая другие управляющие символы C0, суррогатные

блоки и коды, не являющиеся символами. Синтаксис символов формально определён правилом yang-char раздела 14.

Модули и субмодули YANG сохраняются в файлах с использованием кодировки UTF-8 [RFC3629].

Строки в модулях YANG завершаются комбинацией символов перевода строки и возврата каретки. Символ перевода строки, за которым не следует символ возврата каретки, может присутствовать только внутри заключённых в кавычки строк (6.1.3. Кавычки). Отметим, что символы возврата каретки и перевода строки внутри строк, заключённых в кавычки, являются частью этих строк, т. е. многострочные значения в кавычках используют ту же форму завершения строк, что и обычные строки модулей YANG.

6.1. Лексическая маркировка

Модули YANG анализируются как последовательности маркеров. В этом параграфе рассмотрены правила распознавания маркеров во входном потоке. Правила разбора маркеров YANG просты и обеспечивают широкие возможности. Простота обусловлена необходимостью облегчить реализацию синтаксических 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 */.

Отметим, что внутри строк, заключённых в кавычки (6.1.3. Кавычки), указанные последовательности символов никогда не интерпретируются как начало и завершение комментария.

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

Строкой без кавычек считается любая последовательность символов без пробелов, табуляторов, перевода строки или возврата каретки, одинарных или двойных кавычек, точки с запятой (;), фигурных скобок ({ или }) или комментариев (//, /*, */).

Отметим, что любое из ключевых слов может включаться в строку без кавычек.

Внутри строки без кавычек значение каждого символа сохраняется. Это значит, что символ обратной дробной черты (backslash) не имеет какого-либо специального значения в строках без кавычек.

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 \.

Использование других символов после \ **недопустимо**.

Если после строки в кавычках следует символ сложения (+) и другая строка в кавычках, эти строки объединяются в одну (конкатенация). Между строкой в кавычках и знаком + могут включаться пробелы, разрыв строки и комментарии.

Для строк в двойных кавычках перед преобразованием символов с префиксом \ выполняется удаление ненужных пробелов. Конкатенация выполняется после всего.

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 указания 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 и **могут** поддерживать более длинные идентификаторы. Регистр символов в идентификаторах принимается во внимание. Синтаксис идентификаторов формально определён правилом identifier в разделе 14. Идентификаторы могут задаваться в форме строк в кавычках и без кавычек.

6.2.1. Идентификаторы и 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 расширений.
- Все имена функций (feature), определённых в модуле и его субмодулях, используют общее пространство имён функций.
- 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 применять данное определение типа (typedef), а использование другого 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 применять данную группировку (grouping), а использование того же имени для других группировок **недопустимо**.
- Все узлы типа leaf, leaf-list, list, container, choice, rpc, action, notification, anydata и anyxml, определённые (напрямую или с помощью оператора uses) в родительском узле или на верхнем уровне модуля и его субмодулей, 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 (6.1.2. Маркеры).

6.3.1. Расширения языка

Модуль может задавать расширения языка YANG с помощью ключевого слова extension (7.19. Оператор extension). Расширения могут импортироваться другими модулями с помощью оператора import (7.1.5. Оператор import). 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.

Обработка расширения зависит от поддержки этого расширения в анализаторе YANG или наборе инструментов, в которой анализатор встроен. Неподдерживаемые расширения, появляющиеся в модуле YANG как неизвестные операторы (14. Грамматика ABNF для YANG), **можно** игнорировать целиком. Все поддерживаемые расширения **должны** обрабатываться в соответствии со спецификацией.

Следует соблюдать осторожность при определении расширений, чтобы использующие их модули могли применяться и в приложениях, которые не поддерживают расширений.

6.4. Вычисление XPath

Язык YANG опирается на язык путей XML (XPath[1]) 1.0 [XPATH] в качестве нотации для задания связей и зависимостей между узлами. Реализации 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 (6.4. Вычисление XPath). 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 (параграф 3.1 в [XSLT]). Это означает, в частности, что корневой узел может иметь любое число элементов в качестве своих потомков.

Для дерева данных не существует порядка документов. Реализация должна выбрать тот или иной порядок, но этот выбор определяется самой реализацией. Это означает, что выражениям XPath в модулях YANG **не следует** предполагать конкретный порядок документов.

Числовые значения в XPath 1.0 являются действительными числами IEEE 754 [IEEE754-2008] с двойной точностью (параграф 3.5 в [XPATH]). Это значит, что некоторые значения типов int64, uint64 и decimal64 (параграфы 9.2 и 9.3) не могут быть точно представлены в выражениях XPath. Поэтому следует проявлять осторожность при использовании 64-битовых числовых значений в выражениях XPath. В частности, проверки, использующие равенство, могут давать неожиданные результаты.

Для примера рассмотрим выражение

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

Экземпляр листа lxiv, имеющий значение 10,0000000000000001, пройдёт такую проверку.

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 (7.13. Оператор uses). Внутри typedef это пространство зависит от того, где typedef упоминается. Если typedef определён и упоминается в группировке, пространство зависит от того, где группировка применяется (7.13. Оператор uses).
- Библиотека функций - это основная библиотека, определённая в [XPATH], и функции, заданные в разделе 10.
- Набор привязок переменных пуст.

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

Доступное дерево зависит от того, где определён оператор с выражением XPath.

- Если выражение XPath определено в субоператоре для узла данных, который представляет конфигурацию, доступным деревом будут данные в хранилище, где существует контекст узла. Корневой узел в качестве своих потомков имеет все узлы конфигурационных данных верхнего уровня во всех модулях.
- Если выражение XPath определено в субоператоре для узла данных, который представляет состояние, доступным деревом будут все данные на сервере и рабочее хранилище конфигурации. Корневой узел в качестве своих потомков имеет все узлы данных верхнего уровня во всех модулях.
- Если выражение XPath определено в субоператоре оператора notification, доступным деревом будет экземпляр уведомления, все данные на сервере и рабочее хранилище конфигурации. Если уведомление определено на верхнем уровне модуля, корневой узел будет иметь в качестве своих потомков узел, представляющий уведомление, которое будет определено, и все узлы данных верхнего уровня во всех модулях. В противном случае корневой узел в качестве своих потомков имеет все узлы данных верхнего уровня во всех модулях.
- Если выражение XPath определено в субоператоре для input в rpc или action, доступным деревом будет RPC или экземпляр операции, все данные на сервере и рабочее хранилище конфигурации. Корневой узел имеет в качестве своих потомков все узлы данных верхнего уровня во всех модулях. Для RPC корневой узел также имеет узел, представляющий операцию RPC, которая будет определена, как потомок. Узел, представляющий операцию, которая будет определена, имеет в качестве потомков входные параметры операции.
- Если выражение XPath определено в субоператоре для output в rpc или action, доступным деревом будет RPC или экземпляр операции, все данные на сервере и рабочее хранилище конфигурации. Корневой узел имеет в качестве своих потомков все узлы данных верхнего уровня во всех модулях. Для RPC корневой узел также имеет узел, представляющий операцию RPC, которая будет определена, как потомок. Узел, представляющий операцию, которая будет определена, имеет в качестве потомка входные параметры операции.

В доступном дереве существуют все leaf и leaf-list с принятыми по умолчанию значениями (параграфы 7.6.1 и 7.7.2).

[1] XML Path Language.

Если узел, который существует в доступном дереве, имеет контейнер без присутствия (non-presence container) в качестве потомка, контейнер без присутствия существует также в доступном дереве.

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

6.4.1.1. Примеры

Рассмотрим показанный ниже модуль

```
module example-a {
  yang-version 1.1;
  namespace urn:example:a;
  prefix a;
  container a {
    list b {
      key id;
      leaf id {
        type string;
      }
      notification down {
        leaf reason {
          type string;
        }
      }
      action reset {
        input {
          leaf delay {
            type uint32;
          }
        }
        output {
          leaf result {
            type string;
          }
        }
      }
    }
  }
  notification failure {
    leaf b-ref {
      type leafref {
        path "/a/b/id";
      }
    }
  }
}
```

И дерево данных, заданное в XML.

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

Доступным деревом для уведомления down на /a/b[id="2"] будет

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
    <down>
      <reason>error</reason>
    </down>
  </b>
</a>
// здесь возможны другие узлы верхнего уровня
```

Доступным деревом для операции вызова reset на /a/b[id="1"] с параметром when, имеющим значение 10, будет

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
    <reset>
      <delay>10</delay>
    </reset>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
// здесь возможны другие узлы верхнего уровня
```

Доступным деревом для вывода этой операции будет

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
    <reset>
      <result>ok</result>
    </reset>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
// здесь возможны другие узлы верхнего уровня
```

Доступным деревом для уведомления failure может быть

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
<failure>
  <b-ref>2</b-ref>
</failure>
// здесь возможны другие узлы верхнего уровня
```

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 14. Грамматика ABNF для YANG. 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 импортированном модуле.

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." {
  ex: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 служит его идентификатором (6.2. Идентификаторы). Имена модулей, публикуемые в RFC [RFC4844], **должны** выделяться IANA (раздел 14 в [RFC6020]). Частные (фирменные) имена модулей назначаются владеющей модулем организацией без централизованной регистрации. Рекомендации по выбору имён для модулей приведены в параграфе 5.1. Модули и субмодули.

Модуль обычно имеет показанную ниже структуру.

```
module <module-name> {

  // данные 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
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

7.1.2. Оператор yang-version

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 значение 1.1.

Модуль или субмодуль, не включающий оператора yang-version или содержащий оператор с аргументом 1, разработан с использованием языка YANG версии 1, определённой в [RFC6020].

Обработка операторов yang-version со значением версии, отличающимся от 1.1 (определена здесь), выходит за рамки данной спецификации. Любой документ, определяющий более высокий номер версии, должен описать совместимость этой версии с более ранними.

Совместимость YANG версий 1 и 1.1 рассматривается в разделе 12. Сосуществование с YANG версии 1.

7.1.3. Оператор namespace

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

См. также параграф 5.3. Пространства имён XML.

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 для ссылок на содержащиеся в модуле определения (например, if:ifName). Префикс является идентификатором (6.2. Идентификаторы).

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 модуле префикс в качестве префикса пространства имён XML, если это не вызывает конфликтов.

При использовании в операторе import субоператор prefix указывает 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 заданный модулем префикс **следует** использовать при импорте модуля, если это не вызывает конфликтов. При возникновении конфликта, когда импортируются два разных модуля с одинаковыми префиксами, 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, 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 несколько 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 неизвестен.

Можно использовать множество выпусков импортируемого модуля, указав для них разные префиксы.

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`prefix`	7.1.4	1
`reference`	7.21.4	0..1
`revision-date`	7.1.5.1	0..1

7.1.5.1. Оператор revision-date для оператора import

Субоператор revision-date используется в операторе import для указания выпуска импортируемого модуля.

7.1.6. Оператор include

Оператор include делает содержимое субмодуля доступным в родительском модуле. Аргументом оператора служит имя включаемого модуля. В модули можно включать лишь относящиеся к ним субмодули, указанные оператором belongs-to (7.2.2. Оператор belongs-to).

Когда модуль включает в себя субмодуль, он встраивает содержимое субмодуля в свою иерархию узлов.

Для совместимости с YANG версии 1 в субмодули можно включать другие субмодули, относящиеся к тому же модулю, но это не требуется делать в YANG версии 1.1 (5.1. Модули и субмодули).

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 неизвестен.

Недопустимо включение в модуль нескольких выпусков одного субмодуля.

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`reference`	7.21.4	0..1
`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`	7.21.3	0..1
`reference`	7.21.4	0..1

7.1.10. Пример использования

Приведённый ниже пример основан на [RFC6991].

```
module example-system {
  yang-version 1.1;
  namespace "urn:example:system";
  prefix "sys";

  import ietf-yang-types {
    prefix "yang";
    reference "RFC 6991: Common YANG Data Types";
  }

  include example-types;
```

```
    organization "Example Inc.";
    contact
      "Joe L. User

       Example Inc.
       42 Anywhere Drive
       Nowhere, CA 95134
       USA

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

    description
      "Модуль для сущностей, реализующих систему Example.";

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

    // далее следуют 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 (6.2. Идентификаторы).

Имена субмодулей, публикуемые в RFC [RFC4844], **должны** выделяться IANA (раздел 14 в [RFC6020]). Частные (фирменные) имена субмодулей назначаются владеющими модулями организациями без централизованной регистрации. Рекомендации по выбору имён для субмодулей приведены в параграфе 5.1. Модули и субмодули.

Субмодуль обычно имеет показанную ниже структуру.

```
submodule <module-name> {

  <оператор yang-version>

  // идентификация модуля
  <оператор belongs-to>

  // операторы привязок
  <операторы import>

  // 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*
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
belongs-to	7.2.2	1
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
organization	7.1.7	0..1
reference	7.21.4	0..1

revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

7.2.2. 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. Пример использования

```
submodule example-types {
  yang-version 1.1;
  belongs-to "example-system" {
    prefix "sys";
  }

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

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

     Example Inc.
     42 Anywhere Drive
     Nowhere, CA 95134
     USA

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

  description
    "Этот субмодуль определяет базовые типы Example.";
  revision "2007-06-09" {
    description "Исходный 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. Вложенные определения типов и группировки. 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	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.3.2	1
units	7.3.3	0..1

7.3.2. Оператор type

Обязательный оператор type определяет базовый тип, на основе которого создаётся производный (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 (9. Встроенные типы) или производного типа (7.3. Оператор typedef), 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. Встроенные типы.

7.4.1. Субоператоры для type

Субоператор	*Параграф*	*Число элементов*
base	7.18.2	0..n
bit	9.7.4	0..n
enum	9.6.4	0..n
fraction-digits	9.3.4	0..1
length	9.4.4	0..1
path	9.9.2	0..1
pattern	9.4.5	0..n
range	9.2.4	0..1
require-instance	9.9.3	0..1
type	7.4	0..n

7.5. 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 не имеет значения, но имеет список дочерних узлов дерева данных. Дочерние узлы определяются в субоператорах контейнера.

7.5.1. Контейнеры

YANG поддерживает два типа контейнеров - те, которые существуют только для организации иерархии узлов данных, и те, чьё присутствие в дереве данных имеет явный смысл.

В первом случае контейнер не имеет прямого назначения и существует лишь для организации дочерних узлов. В частности, наличие контейнерного узла без дочерних узлов семантически эквивалентно отсутствию контейнерного узла. В YANG этот вариант называется контейнером без присутствия (non-presence container). 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 смысла и 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.

Для данных конфигурации контейнер выступает в качестве конфигурационного элемента и средства организации связанных узлов конфигурации. 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], а также может включать любые связанные с SSH конфигурационные «кнопки», такие как скорость соединения или число попыток.

Оператор presence (7.5.5. Оператор presence) служит для указания смысла наличия контейнера в дереве данных.

7.5.2. Субоператоры для container

Субоператор	*Параграф*	*Число элементов*
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n

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

description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
notification	7.16	0..n
presence	7.5.5	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1

7.5.3. Оператор must

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

При проверке хранилища данных на пригодность, все ограничения must концептуально проверяются один раз для каждого узла в доступном дереве (6.4.1. Контекст XPath).

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

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

- Если оператор must является субоператором notification, узел контекста представляет уведомление в доступном дереве.
- Если оператор must является субоператором input, узел контекста представляет операцию в доступном дереве.
- Если оператор must является субоператором output, узел контекста представляет операцию в доступном дереве.
- В остальных случаях узел контекста является узлом в доступном дереве, для которого оператор must определён.

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

Поскольку все значения листьев в дереве данных концептуально хранятся в канонической форме (9.1. Каноническое представление), любые сравнения XPath также выполняются для канонических значений.

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

7.5.4. Субоператоры для must

Субоператор	*Параграф*	*Число элементов*
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

7.5.4.1. Оператор error-message

Необязательный оператор error-message принимает в качестве аргумента строку. Если оценка ограничений даёт значение false, строка передаётся как <error-message> в сообщении NETCONF <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 сообщении NETCONF <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 ifMTU = 1500' {
    error-message "An Ethernet MTU must be 1500";
  }
  must 'ifType != "atm" or'
     + ' (ifMTU <= 17966 and ifMTU >= 64)' {
    error-message "An ATM MTU must be 64 .. 17966";
  }
}
```

7.5.5. 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 включает 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. Контейнеры.

7.5.6. Операторы дочерних узлов контейнера

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

7.5.7. Правила представления 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 (7.1.3. Оператор namespace).

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.

Любые пробельные символы между субэлементами контейнера не имеют значения и реализация **может** вставлять такие символы между субэлементами.

Если контейнер без присутствия не имеет дочерних узлов, его можно не включать в представление XML.

7.5.8. Операции NETCONF <edit-config>

Контейнеры могут создаваться, удаляться, заменяться и изменяться с помощью операции <edit-config> с атрибутом operation (параграф 7.2 в [RFC6241]) в элементе XML этого контейнера.

Если узел-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 "Enables SSH";
      description
        "Зависящая от службы конфигурация SSH.";
      // 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="urn:example:config">
        <services>
          <ssh nc:operation="delete"/>
        </services>
```

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

7.6. Оператор leaf

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

Узел leaf имеет значение, но не имеет дочерних узлов в дереве данных. Концептуально значение листа в дереве данных всегда имеет каноническую форму (9.1. Каноническое представление).

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 (7.5.1. Контейнеры).

- 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 хотя бы один узел case существует в дереве данных или узел case является принятым по умолчанию для choice и нет узлов от других вариантов case в дереве данных.
- В остальных случаях принятое по умолчанию значение **должно** применяться при наличии предка в дереве данных.

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

Отметим, что принятое по умолчанию значение не применяется, если лист или любой из его предков имеет условие when или выражение if-feature, дающее в результате значение false.

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 этом операторе. 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	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1

7.6.3. Оператор type

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

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 значения **недопустимо** помечать оператором if-feature. Например, ниже представлено неприемлемое определение.

```
leaf color {
  type enumeration {
    enum blue { if-feature blue; }
    ...
  }
  default blue; // недопустимо, поскольку значение enum является условным
}
```

7.6.5. Оператор mandatory

Необязательный оператор mandatory 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 (7.5.1. Контейнеры):

- 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 case;
- в остальных случаях leaf **должен** существовать, если узел-предок существует в дереве данных.

Эти ограничения применяются в соответствии с правилами раздела 8. Ограничения.

7.6.6. Правила представления XML

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

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

Пример представления дан в параграфе 7.6.8. Пример использования.

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 контейнер ssh (7.5.9. Пример использования):

```
leaf port {
  type inet:port-number;
  default 22;
  description
    "Порт, прослушиваемый сервером 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="urn:example: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 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.

Определения используемых по умолчанию значений **недопустимо** помечать операторами if-feature.

Концептуально значения в дереве данных **должны** иметь каноническую форму (9.1. Каноническое представление).

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 сервер может сортировать элементы по своему усмотрению. Строка description для списка может предложить порядок для разработчиков сервера. В языке YANG такой стиль называется

системным упорядочением, 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 сервер этот порядок поддерживает. В YANG такие списки называются упорядоченными пользователем и указываются оператором ordered-by user. Например, порядок, 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 указанным элементом или после него.

Оператор ordered-by описан в параграфе 7.7.7. Оператор ordered-by.

7.7.2. Принятые по умолчанию значения leaf-list

Значения leaf-list, принятые по умолчанию, сервер использует при отсутствии узла leaf-list в дереве данных. Применение этих значений зависит от ближайшего предка leaf-list в дереве схемы, который не является контейнером без присутствия (7.5.1. Контейнеры):

- если в дереве схемы нет таких предков, **должны** использоваться принятые по умолчанию значения;
- в противном случае, если предок является узлом case, принятые по умолчанию значения **должны** применяться, когда хотя бы один узел case присутствует в дереве данных или узел case является принятым по умолчанию вариантом выбора (choice), а других узлов из case в дереве данных нет;
- в остальных случаях принятые по умолчанию значения **должны** применяться, если узел-предок имеется в дереве данных.

В таких случаях говорят, что используются принятые по умолчанию значения.

Отметим, что при наличии у leaf-list или любого из его предков условия when или выражения if-feature со значением false принятые по умолчанию значения не используются.

Когда принятые по умолчанию значение используются, сервер **должен** вести себя так, будто leaf-list присутствует в дереве данных и имеет значения, совпадающие с принятыми по умолчанию.

Если leaf-list имеет один или множество операторов default, принятыми по умолчанию значениями для leaf-list являются значения операторов default, а если порядок leaf-list задан пользователем, эти значения применяются в порядке операторов default. В остальных случаях, если тип leaf-list имеет принятое по умолчанию значение, а leaf-list не имеет операторов min-element со значением не меньше 1, принятым по умолчанию значением leaf-list будет принятое по умолчанию значение одного из экземпляров его типа. Во всех остальных случаях leaf-list не имеет принятого по умолчанию значения.

7.7.3. Субоператоры для leaf-list

Субоператор	*Параграф*	*Число элементов*
`config`	7.21.1	0..1
`default`	7.7.4	0..n
`description`	7.21.3	0..1
`if-feature`	7.20.2	0..n
`max-elements`	7.7.6	0..1
`min-elements`	7.7.5	0..1
`must`	7.5.3	0..n
`ordered-by`	7.7.7	0..1
`reference`	7.21.4	0..1
`status`	7.21.2	0..1
`type`	7.4	1
`units`	7.3.3	0..1
`when`	7.21.5	0..1

7.7.4. Субоператор default для leaf-list

Необязательный оператор default принимает один параметр с принятым по умолчанию значением для leaf-list.

Значение оператора default **должно** быть пригодно для типа, заданного оператором type для leaf-list.

Оператор default **недопустимо** включать для узлов, в которых min-elements имеет значение не меньше 1.

7.7.5. 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, принимается значение 0.

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

- если такого предка нет в дереве схемы, ограничения применяются;

- в противном случае, если предок является узлом case, ограничения применяются при наличии любого другого узла из case;
- в остальных случаях ограничения применяются при наличии узла-предка.

Далее ограничения применяются в соответствии с правилами раздела 8. Ограничения.

7.7.6. Оператор max-elements

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

При отсутствии оператора max-elements используется значение unbounded.

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

7.7.7. 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 применяется system.

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

Дополнительная информация приведена в параграфе 7.7.1. Упорядочение.

7.7.7.1. Системное упорядочение

Элементы списка упорядочиваются системой. Строка оператора description для списка может предлагать порядок сортировки для разработчиков серверных реализаций. Если этого нет, приложение выбирает порядок сортировки по своему усмотрению. Реализациям **следует** применять для одинаковых данных один и тот же порядок, независимо от места расположения данных. Использование определённого порядка позволяет выполнять сравнение с помощью простых средств, таких как diff.

Этот порядок применяется по умолчанию.

7.7.7.2. Упорядочение пользователем

Элементы списка упорядочиваются пользователем. В NETCONF управление порядком осуществляется с помощью специальных атрибутов XML в запросе <edit-config> (7.7.9. Операции NETCONF <edit-config>).

7.7.8. Правила представления XML

Узел leaf-list представляется последовательностью элементов XML. Локальное имя каждого элемента служит идентификатором leaf-list, а в качестве пространства имён используется пространство имён XML для модуля (7.1.3. Оператор namespace).

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 или операции.

Пример представлен в параграфе 7.7.10. Пример использования.

7.7.9. 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 (5.3.1. Пространство имён YANG XML) 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 нового элемента или перемещения существующего.

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, 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.

7.7.10. Пример использования

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

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="urn:example: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="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <cipher nc:operation="create"
                    yang:insert="after"
                    yang:value="3des-cbc">blowfish-cbc</cipher>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.8. 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
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
key	7.8.2	0..1
leaf	7.6	0..n

leaf-list	7.7	0..n
list	7.8	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
notification	7.16	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
unique	7.8.3	0..n
uses	7.13	0..n
when	7.21.5	0..1

7.8.2. 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 нескольких 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, принятые по умолчанию значения в листьях ключа и их типах игнорируются. Все операторы mandatory в листьях ключа игнорируются.

Лист, являющийся частью ключа, может быть любого встроенного или производного типа.

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 14. Грамматика ABNF для YANG.

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 (правило descendant-schema-nodeid в разделе 14. Грамматика ABNF для YANG). 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 или имеют принятые по умолчанию значения. Ограничение применяется в соответствии с правилами раздела 8. Ограничения.

Синтаксис строки unique формально определён правилом unique-arg из раздела 14. Грамматика ABNF для YANG.

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, заданных 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, anydata и anyxmls могут служить для определения дочерних узлов списка.

7.8.5. Правила представления 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 (7.1.3. Оператор namespace). Элемента XML, включающего список целиком, нет.

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 или операции, субэлементы кодируются в том же порядке, как они определены в операторе list. В стальных случаях порядок может быть произвольным.

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 или операции.

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 (5.3.1. Пространство имён YANG XML) 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 нового элемента или перемещения существующего.

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 (9.13. Встроенный тип instance-identifier) для элемента списка.

Если атрибут insert не задан для операции create, 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
    "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="urn:example:config">
        <user nc:operation="create">
          <name>barney</name>
          <type>admin</type>
          <full-name>Barney Rubble</full-name>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

Изменение роли пользователя fred на superuser

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <user>
          <name>fred</name>
          <type>superuser</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

Для упорядоченного пользователем списка

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

  key "first-name surname";

  leaf first-name {
    type string;
  }
  leaf surname {
    type string;
  }
  leaf type {
    type string;
  }
}
```

Приведённый ниже фрагмент будет служить 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="urn:example:config"
           xmlns:ex="urn:example:config">
        <user nc:operation="create"
```

```
                yang:insert="after"
                yang:key="[ex:first-name='fred']
                          [ex:surname='flintstone']">
            <first-name>barney</first-name>
            <surname>rubble</surname>
            <type>admin</type>
          </user>
        </system>
      </config>
    </edit-config>
  </rpc>
```

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

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

7.9. 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 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 и содержит множество дочерних узлов. Одновременно могут существовать узлы не более, чем из одной ветви.

Поскольку в дереве данных в любой момент может действовать только один из вариантов выбора, создание узла в одном из вариантов (case) неявно удаляет все узлы из других вариантов. Если запрос создаёт узел из case, сервер будет удалять все имеющиеся узлы, определённые другими вариантами в этом операторе choice.

7.9.1. Субоператоры для choice

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

7.9.2. 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 anydata, 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 {     // Пример недействителен в YANG
    case a {
      leaf ethernet { ... }
    }
    case b {
      container ethernet { ...}
    }
  }
```

Для сокращения оператор case может быть опущен, если вариант содержит один оператор anydata, 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 (6.5. Идентификатор узла схемы) 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*
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
uses	7.13	0..n
when	7.21.5	0..1

7.9.3. 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 только при рассмотрении операторов default в дочерних узлах вариантов (т. е. принятых по умолчанию значений leaf и leaf-list, а также используемых по умолчанию вариантов вложенных операторов choice). Принятые по умолчанию значения и варианты вложенных choice в потомках заданного по умолчанию варианта используются при отсутствии узлов в любом из вариантов (case).

Недопустимо наличие обязательных (mandatory) узлов (3. Терминология), 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, заданные 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, 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;
        units minutes;
        default 30;
      }
    }
    case daily {
      leaf daily {
        type empty;
      }
      leaf time-of-day {
        type string;
        units 24-hour-clock;
        default "01.00";
      }
    }
    case manual {
      leaf manual {
        type empty;
      }
    }
  }
}
```

7.9.4. 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, предполагается значение false.

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

- если такого предка нет в дереве схемы, ограничение применяется;
- в противном случае, 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. Ограничения.

7.9.5. Правила представления 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 или action. В остальных случаях субэлементы могут представляться в любом порядке.

7.9.6. 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="urn:example:config">
        <protocol>
          <udp nc:operation="create"/>
        </protocol>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.10. Оператор anydata

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

Оператор anydata служит для представления неизвестного набора узлов, который может моделироваться в YANG (исключая anyxml), но для которого модель данных не была известна в момент разработки модуля. Возможно (но не обязательно) получение модели данных для содержимого anydata через протокол сигнализации или иным способом, выходящим за рамки этого документа.

Примером, где оператор anydata может быть полезен, является список полученных уведомлений, для которого конкретные уведомления на момент разработки модуля не известны.

Узел anydata не может быть дополнен (7.17. Оператор augment).

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

Реализация может не знать модели данных, используемой для моделирования конкретного экземпляра узла anydata.

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

7.10.1. Субоператоры для anydata

Субоператор	Параграф	Число элементов
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.10.2. Правила представления XML

Узел anydata представляется в виде элемента XML. Локальное имя элемента является идентификатором anydata, а пространством имён служит пространство имён XML для модуля (7.1.3. Оператор namespace). Значением узла anydata является множество узлов, которые представляются субэлементами XML для элемента anydata.

7.10.3. Операции NETCONF <edit-config>

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

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

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

- Если задана операция merge или replace, узел создаётся при его отсутствии, а в качестве значения устанавливаются субэлементы узла anydata из данных XML для RPC.
- Если задана операция create, узел создаётся при его отсутствии, а в качестве значения устанавливаются субэлементы узла anydata из 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

Ниже приведён пример использования оператора anydata.

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

Приведённый фрагмент является пригодным представлением экземпляра списка.

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

7.11. 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 (7.17. Оператор augment).

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

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

Следует отметить, что в YANG версии 1 оператор anyxml был единственным оператором для моделирования неизвестной иерархии данных. Во многих случаях такие неизвестные данные реально моделировались в YANG, но конкретная модель данных YANG не была известна в момент разработки. В таких случаях **рекомендуется** использовать оператор anydata (7.10. Оператор anydata) вместо anyxml.

7.11.1. Субоператоры для anyxml

Субоператор	Параграф	Число элементов
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.11.2. Правила представления XML

Узел anyxml представляется в виде элемента XML. Локальное имя элемента является идентификатором anyxml, а пространством имён служит пространство имён XML для модуля (7.1.3. Оператор namespace). 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.

7.11.3. 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.

7.11.4. Пример использования

Для оператора anyxml

```
anyxml html-info;
```

ниже приведены два варианта корректного представления одного значения anyxml.

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

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

7.12. 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. Вложенные определения типов и группировки). 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 (7.13. Оператор uses). 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 в рамках 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. Это означает, что расширения, определённые в форме прямых потомков группировки, применяются к самой группировке.

Если группировка определяет узел action или notification в своей иерархии, этот узел не может применяться в каком-либо контексте, например, он не может использоваться в определении rpc (параграфы 7.15 и 7.16).

7.12.1. Субоператоры для grouping

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

7.12.2. Пример использования

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

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

7.13. 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.

7.13.1. Субоператоры для uses

Субоператор	Параграф	Число элементов
augment	7.17	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
refine	7.13.2	0..n
status	7.21.2	0..1
when	7.21.5	0..1

7.13.2. 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, как был определён в группировке.

Строка аргумента представляет собой идентификатор узла-потомка в схеме (6.5. Идентификатор узла схемы).

Оператор позволяет выполнить перечисленные ниже уточнения.

- Узел leaf или choice может получить используемое по умолчанию значение или обновить имеющееся.
- Узел leaf-list может получить набор используемых по умолчанию значений или обновить имеющиеся (т. е. новый набор будет установлен взамен прежнего).
- 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, anydata, anyxml или choice может получить другой оператор mandatory.
- Контейнер может получить оператор presence.
- Узел leaf, leaf-list, list, container, anydata или anyxml может получить дополнительные выражения must.

- Узел leaf-list или list может получить другие операторы min-elements или max-elements.
- Узел leaf, leaf-list, list, container, choice, case, anydata или anyxml может получить дополнительные выражения if-feature.
- Любой узел может получить уточнённые расширения, если те допускают уточнение (7.19. Оператор extension).

7.13.3. Правила представления 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.

7.13.4. Пример использования

Для использования группировки endpoint, определённой в параграфе 7.12.2, при определении сервера HTTP в каком-то другом модуле можно задать

```
import example-system {
  prefix "sys";
}

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

Ниже приведён пример с ошибкой.

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

7.14. Оператор rpc

Оператор rpc служит для определения операций вызова удалённой процедуры (RPC). Он принимает один аргумент в форме идентификатора, за которым следует блок субоператоров, определяющих операцию вызова удалённой процедуры. Аргументом является имя RPC.

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

7.14.1. Субоператоры для rpc

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`grouping`	7.12	0..n
`if-feature`	7.20.2	0..n
`input`	7.14.2	0..1
`output`	7.14.3	0..1
`reference`	7.21.4	0..1
`status`	7.21.2	0..1
`typedef`	7.3	0..n

7.14.2. Оператор input

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

Если лист в дереве input имеет оператор mandatory со значением true, этот лист **должен** присутствовать в вызове RPC.

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

Если узел leaf-list в дереве input имеет одно или несколько принятых по умолчанию значений, сервер **должен** использовать эти значения в случаях, описанных в параграфе 7.7.2. В таких ситуациях сервер **должен** вести себя так, будто узел leaf-list был представлен при вызове RPC с принятыми по умолчанию значениями.

Поскольку дерево input не помещается в какое-либо хранилище данных, все операторы config для узлов этого дерева игнорируются.

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

7.14.2.1. Субоператоры для input

Субоператор	*Параграф*	*Число элементов*
`anydata`	7.10	0..n
`anyxml`	7.11	0..n
`choice`	7.9	0..n
`container`	7.5	0..n
`grouping`	7.12	0..n
`leaf`	7.6	0..n
`leaf-list`	7.7	0..n
`list`	7.8	0..n
`must`	7.5.3	0..n
`typedef`	7.3	0..n
`uses`	7.13	0..n

7.14.3. Оператор output

Необязательный оператор output служит для определения выходных значений 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 RPC.

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

Если узел leaf-list в дереве output имеет одно или несколько принятых по умолчанию значений, клиент **должен** использовать эти значений в случаях, описанных в параграфе 7.7.2. В таких ситуациях клиент **должен** вести себя так, будто в выходе RPC присутствует leaf-list с принятыми по умолчанию значениями.

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

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

7.14.3.1. Субоператоры для output

Субоператор	*Параграф*	*Число элементов*
`anydata`	7.10	0..n
`anyxml`	7.11	0..n
`choice`	7.9	0..n
`container`	7.5	0..n
`grouping`	7.12	0..n
`leaf`	7.6	0..n
`leaf-list`	7.7	0..n
`list`	7.8	0..n
`must`	7.5.3	0..n
`typedef`	7.3	0..n
`uses`	7.13	0..n

7.14.4. Правила представления NETCONF XML

Узел rpc представляется как дочерний элемент XML элемента <rpc> в соответствии с группой подстановки rpcOperation [RFC6241]. Локальное имя элемента является идентификатором rpc, а его пространство имён - пространством имён XML для модуля (7.1.3. Оператор namespace).

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 [RFC6241]. Если выходные параметры присутствуют, они кодируются как дочерние элементы элемента <rpc-reply>, определённого в [RFC6241], с тем же порядком, который был указан в операторе output.

7.14.5. Пример использования

Приведённый ниже пример определяет операцию RPC.

```
module example-rock {
  yang-version 1.1;
  namespace "urn:example: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="urn:example:rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

7.15. Оператор action

Оператор action служит для определения операции, подключённой к конкретному контейнеру или списку. Оператор принимает в качестве аргумента идентификатор узла, за которым следует блок субоператоров с информацией об операции. Аргумент является именем операции (action).

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

Действие **недопустимо** определять в rpc, другом действии или уведомлении (notification), т. е. узлу action **недопустимо** иметь в дереве схемы родительский узел типа rpc, action или notification. Это означает, что будет ошибкой, если группировка будет включать узел action в своей иерархии, которая используется для того, чтобы определить уведомление.

Узлу action **недопустимо** иметь в качестве родителя узел, который является списком без оператора key.

Поскольку операция не может быть определена на верхнем уровне модуля или в операторе case, будет ошибкой, если группировка, содержащая action на вершине своей иерархии, используется на верхнем уровне модуля или в определении case.

Различие между action и rpc заключается в том, что операция привязана к узлу в хранилище данных, а rpc нет. При вызове операции задаётся узел в хранилище данных вместе с именем операции и входными параметрами.

7.15.1. Субоператоры для action

Субоператор	Параграф	Число элементов
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n

7.15.2. Правила представления NETCONF XML

При вызове операции элемент с локальным именем action в пространстве имён urn:ietf:params:xml:ns:yang:1 (5.3.1. Пространство имён YANG XML) представляется в виде дочернего элемента XML для элемента <rpc>, определённого в [RFC6241], как указано группой подстановки rpcOperation в [RFC6241].

Элемент <action> содержит иерархию узлов, указывающую узел в хранилище данных. Он **должен** включать все контейнеры и списки на прямом пути от верхнего уровня до списка или контейнера, включающего операцию (action). Для списков все листья ключей также **должны** включаться. Внутренний (innermost) контейнер или список содержит элемент XML, который передаёт имя заданной операции. Внутри этого элемента входные параметры представляются дочерними элементами XML в том же порядке, какой был задан в операторе input.

В одном элементе <rpc> может быть вызвана только одна операция. Если в <rpc> присутствует более одной операции, сервер должен вернуть <rpc-error> с тегом <error-tag>, имеющим значение bad-element.

Если вызов операции завершился успешно и не было возвращено выходных параметров, <rpc-reply> содержит только элемент <ok/>, определённый в [RFC6241]. Если выходные параметры присутствуют, они кодируются как дочерние элементы элемента <rpc-reply>, определённого в [RFC6241], с тем же порядком, который был задан в операторе output.

7.15.3. Пример использования

Приведённый ниже пример определяет операцию сброса одного сервера (reset) в серверной ферме.

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

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

  list server {
    key name;
    leaf name {
      type string;
    }
    action reset {
      input {
        leaf reset-at {
          type yang:date-and-time;
          mandatory true;
        }
       }
       output {
         leaf reset-finished-at {
           type yang:date-and-time;
           mandatory true;
         }
       }
     }
   }
 }
```

Ниже представлены соответствующие экземпляры XML для полных rpc и rpc-reply.

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

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <reset-finished-at xmlns="urn:example:server-farm">
    2014-07-29T13:42:12Z
  </reset-finished-at>
</rpc-reply>
```

7.16. Оператор notification

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

Уведомление может быть определено на верхнем уровне модуля или связано с конкретным контейнером или списком в дереве схемы.

Уведомления **недопустимо** определять в rpc, action или другом узле notification, т. е. уведомлениям **недопустимо** иметь в качестве узла-предка в дереве схемы rpc, action, notification. Например, это может возникать в случаях, когда группировка (grouping) содержащая уведомление, будет использована при определении rpc.

Уведомлениям **недопустимо** иметь в качестве предка узел списка (list) без оператора key.

Поскольку уведомление не может быть определено в операторе case, использование в варианте (case) группировки, содержащей уведомление на вершине иерархии узлов, будет ошибкой.

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

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

Если leaf-list в дереве уведомления имеет одно или несколько принятых по умолчанию значений, клиент **должен** использовать эти значения в случаях, описанных в параграфе 7.7.2. Принятые по умолчанию значения leaf-list. В таких ситуациях клиент **должен** вести себя так, будто leaf-list с принятыми по умолчанию значениями присутствует в экземпляре notification.

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

7.16.1. Субоператоры для notification

Субоператор	Параграф	Число элементов
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.16.2. Правила представления NETCONF 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 (7.1.3. Оператор namespace).

Когда узел notification определяется как потомок узла данных, элемент <notification>, определённый в [RFC5277], содержит иерархию узлов, указывающих узел данных в хранилище. Она **должна** включать включать все контейнеры и списки на прямом пути от верхнего уровня до списка или контейнера, содержащего уведомление (notification). Для списков все листья ключей также **должны** включаться. Внутренний (innermost) контейнер или список содержит элемент XML, который передаёт имя определённого уведомления.

Дочерние узлы уведомления кодируются как субэлементы элемента XML для узла notification в любом порядке.

7.16.3. Пример использования

Приведённый ниже пример определяет уведомление на верхнем уровне модуля.

```
module example-event {
  yang-version 1.1;
  namespace "urn:example:event";
  prefix "ev";

  notification event {
    leaf event-class {
      type string;
    }
    leaf reporting-entity {
      type instance-identifier;
    }
    leaf severity {
      type string;
    }
  }
}
```

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="urn:example:event">
    <event-class>fault</event-class>
    <reporting-entity>
      /ex:interface[ex:name='Ethernet0']
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

Следующий пример определяет уведомление в узле данных.

```
module example-interface-module {
  yang-version 1.1;
  namespace "urn:example:interface-module";
  prefix "if";

  container interfaces {
    list interface {
      key "name";
      leaf name {
        type string;
      }
      notification interface-enabled {
        leaf by-user {
          type string;
        }
      }
```

```
        }
      }
    }
```

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>
  <interfaces xmlns="urn:example:interface-module">
    <interface>
      <name>eth1</name>
      <interface-enabled>
        <by-user>fred</by-user>
      </interface-enabled>
    </interface>
  </interfaces>
</notification>
```

7.17. 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. Целевой узел должен быть узлом типа container, list, choice, case, input, output или notification. К нему добавляются узлы, указанные с субоператорах, следующих за оператором augment.

Строка дополнения является идентификатором узла схемы (6.5. Идентификатор узла схемы). Если оператор augment относится к верхнему уровню модуля или субмодуля, **должна** применяться абсолютная форма (правило absolute-schema-nodeid в разделе 14. Грамматика ABNF для YANG) идентификатора узла схемы. Если augment является субоператором для uses, **должна** применяться наследуемая (descendant) форма (правило descendant-schema-nodeid в разделе 14).

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 (7.9.2. Оператор case для выбора) форме.

Оператору augment **недопустимо** добавлять несколько узлов с одним именем из одного модуля в один целевой узел.

Если добавляются обязательные узлы (3. Терминология), которые представляют конфигурацию целевого узла в другом модуле, дополнение **должно** быть условным с использованием оператора when. Следует аккуратно определять выражение оператора when, чтобы у клиентов, не знающих о дополняющем модуле, не возникло проблем.

В приведённом ниже примере показано корректное добавление узла interface с mandatory-leaf, поскольку дополнение зависит от поддержки some-new-iftype. Старый клиент не знает этого типа, поэтому он не будет выбирать его и в результате не будет добавлять обязательный узел данных.

```
module example-augment {
  yang-version 1.1;
  namespace "urn:example:augment";
  prefix mymod;

  import ietf-interfaces {
    prefix if;
  }

  identity some-new-iftype {
     base if:interface-type;
  }

  augment "/if:interfaces/if:interface" {
     when 'derived-from-or-self(if:type, "mymod:some-new-iftype")';

     leaf mandatory-leaf {
        mandatory true;
        type string;
     }
  }
}
```

7.17.1. Субоператоры для augment

Субоператор	*Параграф*	*Число элементов*
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n

leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
uses	7.13	0..n
when	7.21.5	0..1

7.17.2. Правила представления 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.

7.17.3. Пример использования

В пространстве имён urn:example:interface-module

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

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

В пространстве имён urn:example:ds0

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

Пример соответствующего экземпляра XML приведён ниже.

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

В другом примере предполагается наличие выбора (choice), определённого в параграфе 7.9.6. Пример использования. 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>
```

7.18. 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 выведено из 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 тип данных identityref (9.10. Встроенный тип identityref).

7.18.1. Субоператоры для identity

Субоператор	*Параграф*	*Число элементов*
`base`	7.18.2	0..n
`description`	7.21.3	0..1
`if-feature`	7.20.2	0..n
`reference`	7.21.4	0..1
`status`	7.21.2	0..1

7.18.2. 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 задано несколько операторов base, новое отождествление является производным от всех базовых.

Если в базовом имени имеется префикс, это говорит о том, что базовое отождествление задано 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.

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

Создание производных отождествлений имеет следующие свойства:

- нерефлексивность (отождествление не выводится из самого себя);
- транзитивность (если отождествление B является производным от A, а C - производным от B, то C также является производным от A).

7.18.3. Пример использования

```
module example-crypto-base {
  yang-version 1.1;
  namespace "urn:example:crypto-base";
  prefix "crypto";

  identity crypto-alg {
    description
      "Базовое отождествление для вывода криптоалгоритмов.";
  }

  identity symmetric-key {
    description
      "Базовое отождествление симметричных криптоалгоритмов.";
    }

  identity public-key {
    description
      " Базовое отождествление алгоритмов с открытым ключом.";
    }
}

module example-des {
  yang-version 1.1;
  namespace "urn:example:des";
  prefix "des";

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

  identity des {
    base "crypto:crypto-alg";
    base "crypto:symmetric-key";
    description "Криптоалгоритм DES.";
  }

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

7.19. 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 этого 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 14. Грамматика ABNF для YANG.

Расширение может разрешать уточнения (7.13.2. Оператор refine) и отклонения (7.20.3.2. Оператор deviate), но механизм для этого выходит за рамки спецификации.

7.19.1. Субоператоры для extension

Субоператор	*Параграф*	*Число элементов*
`argument`	7.19.2	0..1
`description`	7.21.3	0..1
`reference`	7.21.4	0..1
`status`	7.21.2	0..1

7.19.2. Оператор argument

Необязательный оператор argument принимает 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.

7.19.2.1. Субоператор для argument

Субоператор	*Параграф*	*Число элементов*
`yin-element`	7.19.2.2	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 (13. YIN). Если оператор yin-element не задан, по умолчанию предполагается значение false.

7.19.3. Пример использования

Определение расширения может иметь вид

```
module example-extensions {
  yang-version 1.1;
  ...

  extension c-define {
    description
      "Принимает строку name как аргумент, заставляя генератор
       кода использовать данное имя в #define.";
    argument "";
  }
}
```

Использование расширения может иметь вид

```
module example-interfaces {
  yang-version 1.1;

  ...
  import example-extensions {
    prefix "myext";
  }
  ...

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

7.20. Операторы, связанные с соответствием спецификации

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

7.20.1. 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 (7.20.2. Оператор if-feature). 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 серверам 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. Идентификаторы. Это имя используется в операторах if-feature для привязки узлов схемы к данной функции.

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

```
module example-syslog {
  yang-version 1.1;

  ...
  feature local-storage {
    description
      "Это свойство означает, что сервер поддерживает локальное
       хранилище (память, flash, диск), в котором можно хранить
       сообщения syslog.";
  }

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

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

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

Чтобы сервер поддерживал функцию, зависящую от каких-либо других функций (т. е. функцию, включающую один или несколько субоператоров if-feature), этот сервер **должен** поддерживать все связанные зависимостями функции.

7.20.1.1. Субоператоры для feature

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`if-feature`	7.20.2	0..n
`reference`	7.21.4	0..1
`status`	7.21.2	0..1

7.20.2. Оператор if-feature

Оператор if-feature делает родительский оператор условным. Аргументом является логическое выражение с именами функций. В этом выражении имя функции считается истинным (true) тогда и только тогда, когда эта функция поддерживается сервером. Родительский оператор выполняется серверами в том случае, когда логическое выражение истинно (true).

Синтаксис логических выражений if-feature формально определён правилом if-feature-expr в разделе 14. Грамматика ABNF для YANG. Скобки служат для группировки выражений. При вычислении используется порядок действий: группировка (скобки), not, and, or.

Если имя функции в логическом выражении включает префикс, имя этого префикса указывает функцию, определённую в модуле, который был импортирован с данным префиксом, или в локальном модуле, если префикс совпадает с префиксом локального модуля. В остальных случаях функция с совпадающим именем **должна** быть определена в текущем модуле или его субмодуле.

Листу, являющемуся ключом списка, **недопустимо** включать операторы if-feature.

7.20.2.1. Пример использования

В этом примере контейнер target реализуется в том случае, когда функция outbound-tls или outbound-ssh поддерживается сервером.

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

Приведённые ниже примеры эквивалентны.

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

7.20.3. Оператор deviation

Оператор deviation определяет иерархию модуля, которую сервер 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 (6.5. Идентификатор узла схемы).

Отклонения показывают, в чем сервер или класс серверов отклоняется от стандарта. Это означает, что отклонениям **недопустимо** быть частью опубликованного стандарта, поскольку они обеспечивают механизм информирования об отклонении реализации от стандарта.

Серверам настоятельно рекомендуется избегать отклонений и они **должны** использоваться лишь в качестве последнего шанса. Уведомление клиентов об отклонении от стандарта не заменяет собой следование стандартам. Сервер, 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 компонентов для поддержки отдельных частей стандартного модуля. В таких случаях сервер 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 неподдерживаемые части базового модуля или показать использование для их поддержки другого синтаксиса. Делается это с помощью оператора deviation.

После учёта всех анонсированных сервером отклонений (в любом порядке) полученная в результате модель **должна** быть пригодной для использования.

7.20.3.1. Субоператоры для deviation

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`deviate`	7.20.3.2	1 - n
`reference`	7.21.4	0..1

7.20.3.2. Оператор deviate

Оператор deviate указывает отклонение реализации сервера 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 сервере.

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

Если целевой узел имеет свойство, заданное расширением, возможность отклонения для этого свойства определяется расширением (7.19. Оператор extension).

7.20.3.3. Пример использования

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

```
module example-deviations {
  yang-version 1.1;
  namespace "urn:example:deviations";
  prefix md;

  import example-base {
    prefix base;
  }

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

Сервер будет анонсировать оба модуля example-base и example-deviations.

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 являются администраторами
  }
}
```

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";
  ...
}
```

сервер может удалить ограничение must

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

7.21. Субоператоры общего назначения

В этом разделе описаны субоператоры, используемые несколькими другими операторами.

7.21.1. 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.

Если config имеет аргумент false, определение представляет данные состояния. Узлы данных, представляющих состояние, не входят в хранилище конфигурации.

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 не 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.

7.21.2. Оператор status

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

- current означает, что определение является текущим и действует;
- deprecated указывает устаревшее определение, которое, тем не менее, разрешено использовать для обеспечения совместимости с имеющимися реализациями;
- obsolete указывает отменённое определение, которое **не следует** применять в новых реализациях и/или которое 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; // непригоден, поскольку my-type отменен
}
```

7.21.3. Оператор description

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

7.21.4. 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";
  ...
}
```

7.21.5. 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 выражение XPath (6.4. Вычисление XPath), которое служит для формального задания условий. Если выражение XPath даёт значение true для конкретного экземпляра, узел, определённый родительским оператором, будет действительным.

Листьям, являющимся ключами списка, **недопустимо** включать оператор when.

Если лист-ключ определён в группировке, используемой в списке, оператору uses **недопустимо** включать when.

Дополнительная информация приведена в параграфе 8.3.2. Обработка NETCONF <edit-config>.

XPath оценивается в описанном ниже контексте, дополняющем определение параграфа 6.4.1. Контекст XPath.

- Если оператор when является потомком оператора augment, узлом контекста служит 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 такого узла нет, узлом контекста будет корневой узел. Доступное дерево предварительно изменяется в процессе обработки выражения XPath путём удаления всех экземпляров (при наличии) узлов, добавленных оператором augment.
- Если оператор when является потомком оператора uses, choice или case, узлом контекста является ближайший предок узла с оператором when, который является узлом данных. Если такого узла нет, узлом контекста будет корневой узел. Доступное дерево предварительно изменяется в процессе обработки выражения XPath путём удаления всех экземпляров (при наличии) узлов, добавленных оператором uses, choice или case.
- Если оператор when является потомком любого оператора определения, доступное дерево предварительно изменяется в процессе обработки выражения XPath путём замены всех экземпляров узла данных, для которого определён оператор when, фиктивным узлом с тем же именем, но без значения и потомков. Если таких экземпляров нет, фиктивный узел создаётся. Узлом контекста будет этот фиктивный узел.

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

Если выражение XPath ссылается на любой узел, который также имеет связанные операторы when, значения для этих операторов when **должны** проверяться в первую очередь. Кольцевые зависимости между выражениями when **недопустимы**.

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 типа данных, определяемого оператором.

- Ограничение для данных конфигурации **должно** выполняться в дереве данных конфигурации.
- Ограничение для данных состояния **должно** выполняться в дереве данных состояния.
- Ограничение для содержимого уведомления **должно** выполняться в дереве данных уведомления.
- Ограничение для входных параметров RPC или операции (action) **должно** выполняться при вызове RPC или операции.
- Ограничение для выходных параметров RPC или операции (action) **должно** выполняться в откликах RPC или операции.

Приведённые ниже условия должны соблюдаться (true) для всех деревьев данных:

- все значения данных листа (leaf) **должны** соответствовать ограничениям типа для листа, включая ограничения, заданные для этого типа свойствами range, length и pattern;
- все листья листьев-ключей **должны** присутствовать для всех элементов списка;
- узлы **должны** присутствовать не более чем в одном варианте (case) во всех операторах choice;
- не **должно** присутствовать узлов с меткой if-feature, если выражение if-feature сервер определяет как false.
- не **должно** присутствовать узлов с оператором when, если условие в дереве данных дает значение false.

Приведённые ниже условия должны соблюдаться (true) для действительного дерева данных:

- все ограничения must **должны** давать значение true;
- все ограничения на целостность ссылок, заданные оператором path, **должны** соблюдаться;
- все ограничения unique для списков **должны** соблюдаться;
- ограничения оператора mandatory выполняются для листьев (leaf) и выбора (choice), если узел или любой из его предков не имеет условия when или выражения if-feature, дающего значение false;
- ограничения операторов min-elements и max-elements выполняются для list и leaf-list, если узел или любой из его предков не имеет условия when или выражения if-feature, дающего значение false.

Рабочее хранилище конфигурации всегда **должно** быть действительным (пригодным).

8.2. Изменение данных конфигурации

- Если запрос создаёт узлы данных конфигурации в рамках оператора выбора (choice), все существующие узлы из других вариантов выбора удаляются сервером.
- Если запрос изменяет данные конфигурации так, что в каждом узле выражение when даёт результат false, сервер удаляет из дерева данных узлы с выражениями when.

8.3. Модель применения ограничений NETCONF

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

- в процессе анализа содержимого (payload) RPC;
- в процессе выполнения операции <edit-config>;
- 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 сервер реализует.

- Если значение данных leaf не соответствует ограничениям типа листа, включая заданные свойствами range, length и pattern, сервер **должен** ответить сообщением <rpc-error> с тегом <error-tag> invalid-value, а также error-app-tag (7.5.4.2. Оператор error-app-tag) и error-message (7.5.4.1. Оператор error-message), 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 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 сервере, 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 несуществующих данных;
- запросы на создание уже имеющихся данных;
- запросы вставки с параметрами before или after, указывающими отсутствующие элементы;
- запросы на изменение для узлов с оператором when, дающим значение false; в этом случае сервер **должен** передать <error-tag> с unknown-element в сообщении <rpc-error>.

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 «будущим» (candidate), применение ограничений откладывается до вызова операции <commit> или <validate>.

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. Оператор length и 9.4.5. Оператор pattern), а также ограничения диапазонов числовых значений (9.2.4. Оператор range).

Лексическое представление значений некоторых типов применяется в кодировании XML 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 данные в представлении XML, он **должен** использовать каноническую форму, заданную в этом разделе. Другие представления могут использовать иные варианты. Отметим однако, что значения в дереве данных концептуально хранятся в каноническом представлении, как указано здесь. В частности, оценка выражений XPath выполняется с использованием канонической формы, если тип данных имеет такую форму. Если у типа данных нет канонической формы, формат значения **должен** соответствовать лексическому представлению типа данных, но точный формат зависит от реализации.

Некоторые типы имеют лексическое представление, которое зависит от кодирования, например от контекста XML. 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 - целые числа от -128 до 127, включительно;
int16 - целые числа от -32768 до 32767, включительно;
int32 - целые числа от -2147483648 до 2147483647, включительно;
int64 - целые числа от -9223372036854775808 до 9223372036854775807, включительно;
uint8 - целые числа от 0 до 255, включительно;
uint16 - целые числа от 0 до 65535, включительно;
uint32 - целые числа от 0 до 4294967295, включительно;
uint64 - 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 лексическое представление с шестнадцатеричной или восьмеричной записью. 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 (9.2.4. Оператор range).

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[1] или включительной нижней границей, за которой следуют два символа точки (..) и включительное значение верхней границы. Возможно задание нескольких значений или диапазонов, разделённых символом |. При задании нескольких диапазонов они **должны** быть непересекающимися 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).

[1] В этом случае точки и верхняя граница диапазона не указываются. *Прим. перев.*

Синтаксис выражения range формально определяется правилом range-arg в разделе 14. Грамматика ABNF для YANG.

9.2.4.1. Субоператоры для range

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`error-app-tag`	7.5.4.2	0..1
`error-message`	7.5.4.1	0..1
`reference`	7.21.4	0..1

9.2.5. 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, предполагается +.

9.3.2. Каноническая форма

Каноническая форма положительного значения decimal64 не включает знак +. Десятичная точка обязательна. Нули в начале (целой части) 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 (9.2.4. Оператор range).

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, понятные для человека. Допустимо использование символов кодировок Unicode и ISO/IEC 10646 [ISO.10646], включая табуляцию, возврат каретки и перевод строки, но исключая управляющие символы C0, суррогатные блоки и несимволы (noncharacter). Синтаксис string формально определяется правилом yang-string в разделе 14. Грамматика ABNF для YANG.

9.4.1. Лексическое представление

Значение string лексически представляется символьными данными в формате 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 (9.4.4. Оператор length) и pattern (9.4.5. Оператор pattern).

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 встроенных типов string и binary, а также производных от них.

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 нескольких значений или диапазонов, разделённых символом |. При задании нескольких диапазонов они **должны** быть непересекающимися 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 14.

9.4.4.1. Субоператоры для length

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`error-app-tag`	7.5.4.2	0..1
`error-message`	7.5.4.1	0..1
`reference`	7.21.4	0..1

9.4.5. 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 несколько 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.

9.4.5.1. Субоператоры для pattern

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`error-app-tag`	7.5.4.2	0..1
`error-message`	7.5.4.1	0..1
`modifier`	9.4.6	0..1
`reference`	7.21.4	0..1

9.4.6. Оператор modifier

Оператор modifier является необязательным субоператором для pattern и принимает в качестве аргумента строку invert-match. Если в шаблоне присутствует модификатор invert-match, тип ограничивается значениями, не соответствующими шаблону.

9.4.7. Пример использования

Для приведённого ниже определения типа

```
typedef my-base-str-type {
```

```
      type string {
        length "1..255";
      }
    }
```

приемлемо уточнение

```
    type my-base-str-type {
      // допустимое уточнение размера
      length "11 | 42..max"; // 11 | 42..255
    }
```

а приведённое ниже уточнение недопустимо

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

Для определённого ниже типа

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

приведённые ниже строки будут соответствовать ограничению

```
    AB          // допустимо
    9A00        // допустимо
```

а две следующих строки не будут

```
    00ABAB      // недопустимо, слишком длинное
    xx00        // недопустимо, непригодные символы
```

Для типа

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

приведённая ниже строка будет соответствовать ограничению

```
    enabled     // допустимо
```

а следующие строки не соответствуют

```
    10-mbit     // недопустимо, начинается с числа
    xml-element // недопустимо, начинается с недопустимой последовательности xml
```

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 (9.6.4. Оператор enum), 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 (любые символы Unicode со свойством White_Space) 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`	7.21.3	0..1
`if-feature`	7.20.2	0..n
`reference`	7.21.4	0..1
`status`	7.21.2	0..1
`value`	9.6.4.2	0..1

9.6.4.2. Оператор value

Необязательный оператор value используется 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.

Если значение не задано, оно будет назначаться автоматически. В таких случаях при первом определении будет назначаться значение 0, а далее - значение, превышающее на единицу максимальное из использованных значений enum (т. е максимальное значение, заданное явно или неявно до текущего субоператора enum в родительском операторе type).

Отметим, что присутствие if-feature в операторе enum не влияет на автоматически присваиваемые значения.

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

Когда существующий тип enumeration ограничен, оператор value должен **иметь** такое же значение как в базовом типе или отсутствовать и в этом случае значение также совпадёт с базовым типом.

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>
```

Для приведённого определения типа (typedef)

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

следующее уточнение будет допустимо

```
type my-base-enumeration-type {
  // допустимое уточнение enum
  enum yellow;
  enum red {
    value 3;
  }
}
```

а приведённое ниже уточнение недопустимо.

```
type my-base-enumeration-type {
  // недопустимое уточнение enum
  enum yellow {
    value 4; // недопустимое значение
  }
  enum black; // недопустимое добавление нового имени
}
```

В следующем примере показано, как enum можно пометить тегом if-feature, делающим значение допустимым лишь для серверов, анонсирующих соответствующую функцию.

```
type enumeration {
  enum tcp;
  enum ssh {
```

```
        if-feature ssh;
      }
      enum tls {
        if-feature tls;
      }
    }
```

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 назначается имя.

Когда существующий тип bits ограничивается, набор назначенных имён нового типа **должен** быть подмножеством набора имён базового типа. Битовые позиции с такими назначенными именами менять **недопустимо**.

9.7.1. Ограничения

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

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 (9.7.4.2. Оператор position).

9.7.4. Оператор bit

Оператор bit, который служит субоператором для type, **должен** присутствовать для типа bits. Он используется 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 (6.2. Идентификаторы).

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

9.7.4.1. Субоператоры для bit

Субоператор	*Параграф*	*Число элементов*
`description`	7.21.3	0..1
`if-feature`	7.20.2	0..n
`position`	9.7.4.2	0..1
`reference`	7.21.4	0..1
`status`	7.21.2	0..1

9.7.4.2. 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.

Если позиция бита не задана, она назначается автоматически. Если субоператор bit задан в первый раз, присваивается значение 0, далее - значение на 1 больше текущего максимального значения битовой позиции (т. е. битовой позиции, явно или неявно заданной до текущего субоператора bit в родительском операторе type).

Отметим, что присутствие if-feature в операторе enum не влияет на автоматически присваиваемые значения.

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

Когда существующий тип bits ограничен, оператор position **должен** иметь такое же значение как в базовом типе или отсутствовать и в этом случае значение также совпадёт с базовым типом.

9.7.5. Пример использования

Для приведённых ниже typedef и leaf

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

leaf mybits {
  type mybits-type;
  default "auto-sense-speed";
}
```

лексическое представление листа с битовыми полями disable-nagle и ten-mb-only будет иметь вид

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

Ниже показано допустимое уточнение этого типа

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

А следующее уточнение недопустимо

```
type mybits-type {
  // недопустимое уточнение bit
  bit disable-nagle {
    position 2; // недопустимая позиция
  }
  bit hundred-mb-only; // недопустимое добавление нового имени
}
```

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 (9.4.4. Оператор length). Размер значения типа binary указывает число октетов.

9.8.2. Лексическое представление

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

9.8.3. Каноническая форма

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

9.9. Встроенный тип leafref

Встроенный тип leafref ограничен пространством значений некого узла leaf или leaf-list в дереве схемы и может быть дополнительно ограничен соответствующими экземплярами узлов в дереве данных. Субоператор path (9.9.2. Оператор path) служит для указания узла leaf или leaf-list в дереве схемы. Пространством значений ссылающегося узла является пространство значений узла, на который указывает ссылка.

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

Если ссылающийся узел представляет данные конфигурации и свойство require-instance (9.9.3. Оператор require-instance) имеет значение true, указанный ссылкой узел также **должен** представлять конфигурацию.

Недопустимы замкнутые цепочки ссылок leafref.

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

9.9.1. Ограничения

Тип leafref может быть ограничен с помощью оператора require-instance (9.9.3. Оператор require-instance).

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 14. Грамматика ABNF для YANG.

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 перечисленных ниже вариантах контекста в дополнение к приведённому в параграфе 6.4.1. Контекст XPath определению:

- если оператор path определён внутри typedef, узлом контекста будет узел leaf или leaf-list в дереве данных, который ссылается на typedef;

- в противном случае узлом контекста является узел в дереве данных, для которого определён оператор path.

9.9.3. Оператор require-instance

Оператор require-instance, который является субоператором для type, **может** присутствовать для типов instance-identifier и leafref. Он принимает в качестве аргумента значение true или false. Если оператор отсутствует, 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. Ограничения. Если require-instance имеет значение false, это значит, что экземпляр, на который указывает ссылка, **может** существовать в действительных данных.

9.9.4. Лексическое представление

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

9.9.5. Каноническая форма

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

9.9.6. 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 leafref 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="urn:example: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 (7.18. Оператор identity).

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, заданное 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 реализованы на сервере.

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, он указывает на отождествление, определённое в модуле, который был импортирован с этим префиксом, или текущий модуль, если префикс определён в нём или одном из его субмодулей. В остальных случаях отождествление с совпадающим именем **должно** быть определено в текущем модуле или одном из его субмодулей.

Строковое значение узла типа identityref в выражении XPath must или when является полным именем отождествления с присутствием префикса. Если указанное ссылкой отождествление определено в импортированном модуле, префикс в строковом значении является префиксом, определенным в соответствующем операторе import. В остальных случаях префикс в строковом значении является префиксом текущего модуля.

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 7.18.3. Пример использования и приведённым ниже модулем

```
module example-my-crypto {
  yang-version 1.1;
  namespace "urn:example:my-crypto";
  prefix mc;

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

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

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

  container aes-parameters {
    when "../crypto = 'mc:aes'";
    ...
  }
}
```

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

```
<crypto xmlns:des="urn:example:des">des:des3</crypto>
```

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

```
<crypto xmlns:x="urn:example:des">x:des3</crypto>
```

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

```
<crypto xmlns:mc="urn:example: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 (7.4. Оператор type), который используется для задания типа каждого члена объединения. Оператор принимает один аргумент, являющийся строкой с именам типа. В объединение могут входить встроенные и производные типы.

При генерации представления XML значение кодируется в соответствии с типом члена объединения, к которому относится значение. При интерпретации XML значение проверяется на пригодность в соответствии с каждым типом объединения в порядке, заданном операторами type, пока не будет найдено совпадение. Совпадающий тип будет типом значения для узла, проходящего проверку, и представление интерпретируется в соответствии с правилами для этого типа.

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

9.12.1. Ограничения

Тип union не может иметь ограничений. Однако каждый из входящих в объединение типов может иметь ограничения, на основе правил, заданных 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.

9.12.4. Пример использования

Ниже приведён пример объединения (union) типов int32 и enumeration.

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

Следует соблюдать осторожность в тех случаях, когда член объединения имеет тип leafref, в котором свойство require-instance (9.9.3. Оператор require-instance) имеет значение true. Если лист такого типа указывает существующий экземпляр, значение leaf должно заново проверяться на пригодность при удалении указанного экземпляра. Например, в приведённом ниже определении

```
list filter {
  key name;
  leaf name {
    type string;
  }
  ...
}

leaf outbound-filter {
  type union {
    type leafref {
      path "/filter/name";
    }
    type enumeration {
      enum default-filter;
    }
  }
}
```

предполагается, что в в списке filter существует элемент с именем http и лист outbound-filter имеет это значение

```
<filter>
  <name>http</name>
</filter>
<outbound-filter>http</outbound-filter>
```

Если фильтр http будет удалён, значение листа outbound-filter не будет соответствовать leafref и проверяется следующий тип в объединении. Текущее значение (http) не относится к типу enumeration, поэтому результирующая конфигурация будет недействительной.

Если вторым типом объединения будет string вместо enumeration, текущее значение будет соответствовать ему и результирующая конфигурация станет пригодной.

9.13. Встроенный тип instance-identifier

Встроенный тип instance-identifier служит для однозначного указания конкретного экземпляра узла в дереве данных.

Синтаксис для instance-identifier является подмножеством сокращённого синтаксиса XPath и формально определён правилом instance-identifier в разделе 14. Грамматика ABNF для YANG. 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 ключ имеет тип empty, он представляется пустой строкой ("").

Если лист типа instance-identifier представляет данные конфигурации и свойство require-instance (9.9.3. Оператор require-instance) 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 или leaf-list с принятыми по умолчанию значениями (параграфы 7.6.1 и 7.7.2), для того, чтобы данные были пригодны. Это ограничение применяется в соответствии с правилами раздела 8. Ограничения.

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

9.13.1. Ограничения

Тип instance-identifier может быть ограничен с помощью оператора require-instance (9.9.3. Оператор require-instance).

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 элемента списка */
/ex:system/ex:user[ex:name='fred']

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

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

/* instance-identifier для элемента списка, где второй ключ
   (enabled) имеет тип empty */
/ex:system/ex:service[ex:name='foo'][ex:enabled='']

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

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

10. Функции XPath

Этот документ определяет две базовых функции XPath и пять функций XPath для конкретных типов YANG. Подписи функций заданы с использованием синтаксиса, применяемого в [XPATH].

10.1. Функция для набора узлов

10.1.1. current()

```
node-set current()
```

Функция current() не принимает входных параметров и возвращает набор с узлом начального контекста в качестве единственного члена.

10.1.1.1. Пример использования

Для списка

```
list interface {
  key "name";
  ...
  leaf enabled {
    type boolean;
  }
  ...
}
```

показанный ниже лист (leaf) определяет выражение must, которое обеспечивает активность (enable) указанного интерфейса

```
leaf outgoing-interface {
  type leafref {
    path "/interface/name";
  }
  must '/interface[name=current()]/enabled = "true"';
}
```

10.2. Функция для строк

10.2.1. re-match()

```
boolean re-match(string subject, string pattern)
```

Функция re-match() возвращает значение true, если строка subject соответствует регулярному выражению pattern. В остальных случаях возвращается значение false.

Функция re-match() проверяет соответствие строки заданному регулярному выражению, которое является регулярным выражением XML Schema [XSD-TYPES]. Отметим, что это включает неявное закрепление регулярного выражения в начале и в конце.

10.2.1.1. Пример использования

Выражение

```
re-match("1.22.333", "\d{1,3}\.\d{1,3}\.\d{1,3}")
```

возвращает значение true.

Для подсчёта всех логических интерфейсов с именами eth0.<number> можно задать

```
count(/interface[re-match(name, "eth0\.\d+")])
```

10.3. Функция для типов leafref и instance-identifier

10.3.1. deref()

```
node-set deref(node-set nodes)
```

Функция deref() следует по ссылке, указанной первым узлом из числа заданных параметром nodes, и возвращает узлы, на которые он ссылается.

Если первый узел имеет тип instance-identifier, функция возвращает набор, содержащий единственный узел, на который ссылается идентификатор экземпляра, при наличии такого узла. Если такого узла нет, возвращается пустой набор.

Если первый узел имеет тип leafref, функция возвращает набор, содержащий узлы, на которые ссылается leafref. В частности, набор включает узлы, выбранные субоператором path (9.9.2. Оператор path) для leafref, которые имеет такое же значение, как первый узел аргумента.

Если первый узел имеет другой тип, функция возвращает пустой набор.

10.3.1.1. Пример использования

```
list interface {
  key "name type";
  leaf name { ... }
  leaf type { ... }
  leaf enabled {
    type boolean;
  }
  ...
}

container mgmt-interface {
  leaf name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf type {
    type leafref {
      path "/interface[name=current()/../name]/type";
    }
    must 'deref(.)/../enabled = "true"' {
      error-message
        "The management interface cannot be disabled.";
    }
  }
}
```

10.4. Функции для типа identityref

10.4.1. derived-from()

```
boolean derived-from(node-set nodes, string identity)
```

Функция derived-from() возвращает значение true, если любой из узлов, заданных аргументом nodes, является узлом типа identityref и его значением является отождествление, выведенное (7.18.2. Оператор base) из заданного аргументом identity. В остальных случаях функция возвращает false.

Параметр identity является строкой, соответствующей правилу identifier-ref в разделе 14. Грамматика ABNF для YANG. Если отождествление имеет префикс, это говорит об отождествлении, определённом в модуле, который импортирован с этим префиксом, или в локальном модуле, если это его префикс. Если префикс отсутствует, параметр identity указывает отождествление, определённое в текущем модуле или его субмодулях.

10.4.1.1. Пример использования

```
module example-interface {
  yang-version 1.1;

  ...
  identity interface-type;

  identity ethernet {
    base interface-type;
  }

  identity fast-ethernet {
    base ethernet;
  }

  identity gigabit-ethernet {
    base ethernet;
  }

  list interface {
    key name;
    ...
    leaf type {
      type identityref {
        base interface-type;
      }
    }
    ...
  }

  augment "/interface" {
    when 'derived-from(type, "exif:ethernet")';
    // базовые определения Ethernet
  }
  ...
}
```

10.4.2. derived-from-or-self()

```
boolean derived-from-or-self(node-set nodes, string identity)
```

Функция derived-from-or-self() возвращает значение true, если любой из узлов, заданных аргументом nodes, является узлом типа identityref и его значением является отождествление, которое совпадает с отождествлением, заданным аргументом identity, или порождено от такого отождествления (7.18.2. Оператор base). Иначе возвращается false.

Параметром identity является строка, соответствующая правилу identifier-ref в разделе 14. Грамматика ABNF для YANG. Если отождествление имеет префикс, он указывает, что отождествление определено в модуле, импортированном с данным префиксом, или в локальном модуле, если префикс совпадает с префиксом локального модуля. Если префикса нет, аргумент identity указывает отождествление, определённое в текущем модуле или включённом в него субмодуле.

10.4.2.1. Пример использования

Модуль, определённый в параграфе 10.4.1.1 может также иметь

```
augment "/interface" {
  when 'derived-from-or-self(type, "exif:fast-ethernet");
  // определения FastEthernet
}
```

10.5. Функции для типа enumeration

10.5.1. enum-value()

```
number enum-value(node-set nodes)
```

Функция enum-value() проверяет, является ли первый узел из числа указанных аргументом nodes узлом типа enumeration и возвращает его целочисленное значение enum. Если параметр nodes пуст или первый узел из nodes не относится к типу enumeration, функция возвращает NaN (не число).

10.5.1.1. Пример использования

Для модели данных

```
list alarm {
```

```
    ...
    leaf severity {
      type enumeration {
        enum cleared {
          value 1;
        }
        enum indeterminate {
          value 2;
        }
        enum minor {
          value 3;
        }
        enum warning {
          value 4;
        }
        enum major {
          value 5;
        }
        enum critical {
          value 6;
        }
      }
    }
  }
```

приведённое ниже выражение XPath выбирает только сигналы с уровнем важности major и выше

```
/alarm[enum-value(severity) >= 5]
```

10.6. Функции для типа bits

10.6.1. bit-is-set()

```
boolean bit-is-set(node-set nodes, string bit-name)
```

Функция bit-is-set() возвращает значение true, если первый из узлов, указанных аргументом nodes имеет тип bits и флаг bit-name в нем установлен (1). Иначе возвращается значение false.

10.6.1.1. Пример использования

Если интерфейс описывается узлом флагов flags

```
leaf flags {
  type bits {
    bit UP;
    bit PROMISCUOUS
    bit DISABLED;
  }
}
```

приведённое ниже выражение XPath может служить для выбора активных интерфейсов (флаг UP установлен)

```
/interface[bit-is-set(flags, "UP")]
```

11. 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 (7.1.9. Оператор revision), 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 метаданных, включая organization и contact (параграфы 7.1.7. Оператор organization и 7.1.8. Оператор contact).

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. Отметим, что вставка нового элемента перед существующим или изменение порядка имеющихся элементов будут приводить к смене значений существующих элементов, если числовые значения для них не были назначены явно.
- Для типа bits могут быть добавлены новые флаги с сохранением позиций имеющихся флагов. Отметим, что вставка нового флага перед имеющимся или изменение порядка существующих флагов будут менять позиции флагов, если эти позиции не задаются явно.
- 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.
- Оператор base может быть добавлен в identity.
- Оператор base может быть удалён из типа identityref, пока в том остаётся хотя бы один оператор base.
- 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 (3. Терминология) 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 (3. Терминология).
- Оператор if-feature может быть удалён, если его узел не является обязательным (3. Терминология).
- 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. Новые операторы определения данных могут размещаться в любом месте имеющейся последовательности субоператоров.

12. Сосуществование с YANG версии 1

Модулю YANG версии 1.1 **недопустимо** включать субмодуль YANG версии 1, а модулю YANG версии 1 **недопустимо** включать субмодуль YANG версии 1.1.

Модулю или субмодулю YANG версии 1 **недопустимо** импортировать модуль YANG версии 1.1 по выпуску.

Модуль или субмодуль YANG версии 1.1 **может** импортировать модуль YANG версии 1 по выпуску.

Если модуль A (YANG версии 1) импортирует модуль B без указания выпуска и модуль B обновлён до YANG 1.1, сервер **может** одновременно реализовать оба модуля (A и B). В таких случаях сервер NETCONF **должен** анонсировать оба модуля, используя правила, определённые в параграфе 5.6.4. Анонсирование информации о соответствии в NETCONF, также ему **следует** анонсировать модуль A и последний выпуск модуля B, в котором ещё указан YANG версии 1, в соответствии с правилами [RFC6020].

Это правило создано для того, чтобы позволить имеющимся реализациям модулей YANG версии 1 применяться вместе с модулями YANG версии 1.1. Без этого правила обновление одного модуля до YANG версии 1.1 будет создавать каскадный эффект для модулей, которые его импортируют, требуя от них обновления до версии YANG 1.1.

13. 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 такие инструменты, как XSLT и валидаторы XML.

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

13.1. 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 (параграф 7.19) 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 элементом, 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*
action	name	false
anydata	name	false
anyxml	name	false
argument	name	false
augment	target-node	false
base	name	false
belongs-to	module	false
bit	name	false
case	name	false
choice	name	false
config	value	false
contact	text	true
container	name	false
default	value	false
description	text	true
deviate	value	false
deviation	target-node	false
enum	name	false
error-app-tag	value	false
error-message	value	true
extension	name	false
feature	name	false
fraction-digits	value	false
grouping	name	false
identity	name	false
if-feature	name	false
import	module	false
include	module	false
input	<no argument>	n/a
key	value	false
leaf	name	false
leaf-list	name	false
length	value	false
list	name	false
mandatory	value	false
max-elements	value	false
min-elements	value	false
modifier	value	false
module	name	false
must	condition	false
namespace	uri	false
notification	name	false
ordered-by	value	false
organization	text	true
output	<no argument>	n/a
path	value	false
pattern	value	false
position	value	false
prefix	value	false
presence	value	false
range	value	false
reference	text	true
refine	target-node	false
require-instance	value	false
revision	date	false
revision-date	date	false

rpc	name	false
status	value	false
submodule	name	false
type	name	false
typedef	name	false
unique	tag	false
units	name	false
uses	name	false
value	value	false
when	condition	false
yang-version	value	false
yin-element	value	false

13.1.1. Пример использования

Приведённый ниже модуль YANG

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

  import example-extensions {
    prefix "myext";
  }

  list interface {
    key "name";
    leaf name {
      type string;
    }

    leaf mtu {
      type uint32;
      description "The MTU of the interface.";
      myext:c-define "MY_MTU";
    }
  }
}
```

где расширение c-define определено в параграфе 7.19.3. Пример использования, преобразуется в YIN вида

```
<module name="example-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:foo="urn:example:foo"
        xmlns:myext="urn:example:extensions">

  <namespace uri="urn:example:foo"/>
  <prefix value="foo"/>

  <import module="example-extensions">
    <prefix value="myext"/>
  </import>

  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

14. Грамматика ABNF для YANG

В YANG почти все операторы являются неупорядоченными. Грамматика ABNF [RFC5234] [RFC7405] определяет канонический порядок. Для удобочитаемости модуля **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 присутствовать в любом порядке
                      yang-version-stmt
                      namespace-stmt
                      prefix-stmt

submodule-header-stmts = ;; эти операторы могут указываться в любом порядке
                      yang-version-stmt
                      belongs-to-stmt

meta-stmts          = ;; эти операторы могут присутствовать в любом порядке
                      [organization-stmt]
                      [contact-stmt]
                      [description-stmt]
                      [reference-stmt]

linkage-stmts       = ;; эти операторы могут присутствовать в любом порядке
                      *import-stmt
                      *include-stmt

revision-stmts      = *revision-stmt

body-stmts          = *(extension-stmt /
                        feature-stmt /
                        identity-stmt /
                        typedef-stmt /
                        grouping-stmt /
                        data-def-stmt /
                        augment-stmt /
                        rpc-stmt /
                        notification-stmt /
                        deviation-stmt)

data-def-stmt       = container-stmt /
                      leaf-stmt /
                      leaf-list-stmt /
                      list-stmt /
                      choice-stmt /
                      anydata-stmt /
                      anyxml-stmt /
                      uses-stmt

yang-version-stmt   = yang-version-keyword sep yang-version-arg-str
                      stmtend

yang-version-arg-str = < a string that matches the rule >
                       < yang-version-arg >

yang-version-arg    = "1.1"

import-stmt         = import-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          prefix-stmt
                          [revision-date-stmt]
                          [description-stmt]
                          [reference-stmt]
                      "}" stmtsep

include-stmt        = include-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [revision-date-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

namespace-stmt      = namespace-keyword sep uri-str stmtend

uri-str             = < a string that matches the rule >
                      < URI in RFC 3986 >

prefix-stmt         = prefix-keyword sep prefix-arg-str stmtend

belongs-to-stmt     = belongs-to-keyword sep identifier-arg-str
```

```
                        optsep
                        "{" stmtsep
                            prefix-stmt
                        "}" stmtsep

organization-stmt   = organization-keyword sep string stmtend

contact-stmt        = contact-keyword sep string stmtend

description-stmt    = description-keyword sep string stmtend

reference-stmt      = reference-keyword sep string stmtend

units-stmt          = units-keyword sep string stmtend

revision-stmt       = revision-keyword sep revision-date optsep
                        (";" /
                         "{" stmtsep
                             ;; эти операторы могут присутствовать в любом порядке
                             [description-stmt]
                             [reference-stmt]
                         "}") stmtsep

revision-date       = date-arg-str

revision-date-stmt  = revision-date-keyword sep revision-date stmtend

extension-stmt      = extension-keyword sep identifier-arg-str optsep
                        (";" /
                         "{" stmtsep
                             ;; эти операторы могут присутствовать в любом порядке
                             [argument-stmt]
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                         "}") stmtsep

argument-stmt       = argument-keyword sep identifier-arg-str optsep
                        (";" /
                         "{" stmtsep
                             [yin-element-stmt]
                         "}") stmtsep

yin-element-stmt    = yin-element-keyword sep yin-element-arg-str
                        stmtend

yin-element-arg-str = < a string that matches the rule >
                        < yin-element-arg >

yin-element-arg     = true-keyword / false-keyword

identity-stmt       = identity-keyword sep identifier-arg-str optsep
                        (";" /
                         "{" stmtsep
                             ;; эти операторы могут присутствовать в любом порядке
                             *if-feature-stmt
                             *base-stmt
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                         "}") stmtsep

base-stmt           = base-keyword sep identifier-ref-arg-str
                        stmtend

feature-stmt        = feature-keyword sep identifier-arg-str optsep
                        (";" /
                         "{" stmtsep
                             ;; эти операторы могут присутствовать в любом порядке
                             *if-feature-stmt
                             [status-stmt]
                             [description-stmt]
                             [reference-stmt]
                         "}") stmtsep

if-feature-stmt     = if-feature-keyword sep if-feature-expr-str
                        stmtend

if-feature-expr-str = < a string that matches the rule >
                        < if-feature-expr >

if-feature-expr     = if-feature-term
                          [sep or-keyword sep if-feature-expr]

if-feature-term     = if-feature-factor
                          [sep and-keyword sep if-feature-term]
```

```
if-feature-factor   = not-keyword sep if-feature-factor /
                      "(" optsep if-feature-expr optsep ")" /
                      identifier-ref-arg

typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          type-stmt
                          [units-stmt]
                          [default-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                       "}" stmtsep

type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                      (";" /
                       "{" stmtsep
                           [type-body-stmts]
                       "}") stmtsep

type-body-stmts     = numerical-restrictions /
                      decimal64-specification /
                      string-restrictions /
                      enum-specification /
                      leafref-specification /
                      identityref-specification /
                      instance-identifier-specification /
                      bits-specification /
                      union-specification /
                      binary-specification

numerical-restrictions = [range-stmt]

range-stmt          = range-keyword sep range-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

decimal64-specification = ;; эти операторы могут присутствовать в любом порядке
                          fraction-digits-stmt
                          [range-stmt]

fraction-digits-stmt = fraction-digits-keyword sep
                       fraction-digits-arg-str stmtend

fraction-digits-arg-str = < a string that matches the rule >
                          < fraction-digits-arg >

fraction-digits-arg = ("1" ["0" / "1" / "2" / "3" / "4" /
                            "5" / "6" / "7" / "8"])
                      / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

string-restrictions = ;; эти операторы могут присутствовать в любом порядке
                      [length-stmt]
                      *pattern-stmt

length-stmt         = length-keyword sep length-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

pattern-stmt        = pattern-keyword sep string optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [modifier-stmt]
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

modifier-stmt       = modifier-keyword sep modifier-arg-str stmtend
```

```
modifier-arg-str    = < a string that matches the rule >
                      < modifier-arg >

modifier-arg        = invert-match-keyword

default-stmt        = default-keyword sep string stmtend

enum-specification  = 1*enum-stmt

enum-stmt           = enum-keyword sep string optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           *if-feature-stmt
                           [value-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

leafref-specification = ;; эти операторы могут присутствовать в любом порядке
                      path-stmt
                      [require-instance-stmt]

path-stmt           = path-keyword sep path-arg-str stmtend

require-instance-stmt = require-instance-keyword sep
                         require-instance-arg-str stmtend

require-instance-arg-str = < a string that matches the rule >
                           < require-instance-arg >

require-instance-arg = true-keyword / false-keyword

instance-identifier-specification =
                      [require-instance-stmt]

identityref-specification = 1*base-stmt

union-specification = 1*type-stmt

binary-specification = [length-stmt]

bits-specification  = 1*bit-stmt

bit-stmt            = bit-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           *if-feature-stmt
                           [position-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

position-stmt       = position-keyword sep
                      position-value-arg-str stmtend

position-value-arg-str = < a string that matches the rule >
                         < position-value-arg >

position-value-arg  = non-negative-integer-value

status-stmt         = status-keyword sep status-arg-str stmtend

status-arg-str      = < a string that matches the rule >
                      < status-arg >

status-arg          = current-keyword /
                      obsolete-keyword /
                      deprecated-keyword

config-stmt         = config-keyword sep
                      config-arg-str stmtend

config-arg-str      = < a string that matches the rule >
                      < config-arg >

config-arg          = true-keyword / false-keyword

mandatory-stmt      = mandatory-keyword sep
                      mandatory-arg-str stmtend

mandatory-arg-str   = < a string that matches the rule >
                      < mandatory-arg >
```

```
mandatory-arg       = true-keyword / false-keyword

presence-stmt       = presence-keyword sep string stmtend

ordered-by-stmt     = ordered-by-keyword sep
                      ordered-by-arg-str stmtend

ordered-by-arg-str  = < a string that matches the rule >
                      < ordered-by-arg >

ordered-by-arg      = user-keyword / system-keyword

must-stmt           = must-keyword sep string optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

error-message-stmt  = error-message-keyword sep string stmtend

error-app-tag-stmt  = error-app-tag-keyword sep string stmtend

min-elements-stmt   = min-elements-keyword sep
                      min-value-arg-str stmtend

min-value-arg-str   = < a string that matches the rule >
                      < min-value-arg >

min-value-arg       = non-negative-integer-value

max-elements-stmt   = max-elements-keyword sep
                      max-value-arg-str stmtend

max-value-arg-str   = < a string that matches the rule >
                      < max-value-arg >

max-value-arg       = unbounded-keyword /
                      positive-integer-value

value-stmt          = value-keyword sep integer-value-str stmtend

integer-value-str   = < a string that matches the rule >
                      < integer-value >

grouping-stmt       = grouping-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                           *action-stmt
                           *notification-stmt
                        "}") stmtsep

container-stmt      = container-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [presence-stmt]
                           [config-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           *data-def-stmt
                           *action-stmt
                           *notification-stmt
                        "}") stmtsep

leaf-stmt           = leaf-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          [when-stmt]
                          *if-feature-stmt
                          type-stmt
```

```
                          [units-stmt]
                          *must-stmt
                          [default-stmt]
                          [config-stmt]
                          [mandatory-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                        "}" stmtsep

leaf-list-stmt      = leaf-list-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          [when-stmt]
                          *if-feature-stmt
                          type-stmt stmtsep
                          [units-stmt]
                          *must-stmt
                          *default-stmt
                          [config-stmt]
                          [min-elements-stmt]
                          [max-elements-stmt]
                          [ordered-by-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                        "}" stmtsep

list-stmt           = list-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          [when-stmt]
                          *if-feature-stmt
                          *must-stmt
                          [key-stmt]
                          *unique-stmt
                          [config-stmt]
                          [min-elements-stmt]
                          [max-elements-stmt]
                          [ordered-by-stmt]
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                          *(typedef-stmt / grouping-stmt)
                          1*data-def-stmt
                          *action-stmt
                          *notification-stmt
                        "}" stmtsep

key-stmt            = key-keyword sep key-arg-str stmtend

key-arg-str         = < a string that matches the rule >
                      < key-arg >

key-arg             = node-identifier *(sep node-identifier)

unique-stmt         = unique-keyword sep unique-arg-str stmtend

unique-arg-str      = < a string that matches the rule >
                      < unique-arg >

unique-arg          = descendant-schema-nodeid
                      *(sep descendant-schema-nodeid)

choice-stmt         = choice-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           [default-stmt]
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(short-case-stmt / case-stmt)
                        "}") stmtsep

short-case-stmt     = choice-stmt /
                      container-stmt /
                      leaf-stmt /
                      leaf-list-stmt /
                      list-stmt /
                      anydata-stmt /
                      anyxml-stmt
```

```
case-stmt           = case-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *data-def-stmt
                       "}") stmtsep

anydata-stmt        = anydata-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

anyxml-stmt         = anyxml-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           *must-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *refine-stmt
                           *uses-augment-stmt
                       "}") stmtsep

refine-stmt         = refine-keyword sep refine-arg-str optsep
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           *if-feature-stmt
                           *must-stmt
                           [presence-stmt]
                           *default-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [min-elements-stmt]
                           [max-elements-stmt]
                           [description-stmt]
                           [reference-stmt]
                         "}" stmtsep

refine-arg-str      = < a string that matches the rule >
                      < refine-arg >

refine-arg          = descendant-schema-nodeid

uses-augment-stmt   = augment-keyword sep uses-augment-arg-str optsep
                      "{" stmtsep
                          ;; эти операторы могут присутствовать в любом порядке
                          [when-stmt]
                          *if-feature-stmt
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                          1*(data-def-stmt / case-stmt /
                             action-stmt / notification-stmt)
                       "}" stmtsep
```

```
uses-augment-arg-str = < a string that matches the rule >
                       < uses-augment-arg >

uses-augment-arg     = descendant-schema-nodeid

augment-stmt         = augment-keyword sep augment-arg-str optsep
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           [when-stmt]
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           1*(data-def-stmt / case-stmt /
                              action-stmt / notification-stmt)
                        "}" stmtsep

augment-arg-str      = < a string that matches the rule >
                       < augment-arg >

augment-arg          = absolute-schema-nodeid

when-stmt            = when-keyword sep string optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут присутствовать в любом порядке
                            [description-stmt]
                            [reference-stmt]
                         "}") stmtsep

rpc-stmt             = rpc-keyword sep identifier-arg-str optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут присутствовать в любом порядке
                            *if-feature-stmt
                            [status-stmt]
                            [description-stmt]
                            [reference-stmt]
                            *(typedef-stmt / grouping-stmt)
                            [input-stmt]
                            [output-stmt]
                        "}") stmtsep

action-stmt          = action-keyword sep identifier-arg-str optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут присутствовать в любом порядке
                            *if-feature-stmt
                            [status-stmt]
                            [description-stmt]
                            [reference-stmt]
                            *(typedef-stmt / grouping-stmt)
                            [input-stmt]
                            [output-stmt]
                        "}") stmtsep

input-stmt           = input-keyword optsep
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           *must-stmt
                           *(typedef-stmt / grouping-stmt)
                           1*data-def-stmt
                       "}" stmtsep

output-stmt          = output-keyword optsep
                       "{" stmtsep
                           ;; эти операторы могут присутствовать в любом порядке
                           *must-stmt
                           *(typedef-stmt / grouping-stmt)
                           1*data-def-stmt
                       "}" stmtsep

notification-stmt    = notification-keyword sep
                       identifier-arg-str optsep
                       (";" /
                        "{" stmtsep
                            ;; эти операторы могут присутствовать в любом порядке
                            *if-feature-stmt
                            *must-stmt
                            [status-stmt]
                            [description-stmt]
                            [reference-stmt]
                            *(typedef-stmt / grouping-stmt)
                            *data-def-stmt
                        "}") stmtsep
```

```
   deviation-stmt      = deviation-keyword sep
                         deviation-arg-str optsep
                         "{" stmtsep
                             ;; эти операторы могут присутствовать в любом порядке
                             [description-stmt]
                             [reference-stmt]
                             (deviate-not-supported-stmt /
                               1*(deviate-add-stmt /
                                  deviate-replace-stmt /
                                  deviate-delete-stmt))
                         "}" stmtsep

   deviation-arg-str   = < a string that matches the rule >
                         < deviation-arg >

   deviation-arg       = absolute-schema-nodeid

   deviate-not-supported-stmt =
                         deviate-keyword sep
                         not-supported-keyword-str stmtend

   deviate-add-stmt    = deviate-keyword sep add-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; эти операторы могут присутствовать в любом порядке
                              [units-stmt]
                              *must-stmt
                              *unique-stmt
                              *default-stmt
                              [config-stmt]
                              [mandatory-stmt]
                              [min-elements-stmt]
                              [max-elements-stmt]
                          "}") stmtsep

   deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; эти операторы могут присутствовать в любом порядке
                              [units-stmt]
                              *must-stmt
                              *unique-stmt
                              *default-stmt
                          "}") stmtsep

   deviate-replace-stmt = deviate-keyword sep replace-keyword-str optsep
                         (";" /
                          "{" stmtsep
                              ;; эти операторы могут присутствовать в любом порядке
                              [type-stmt]
                              [units-stmt]
                              [default-stmt]
                              [config-stmt]
                              [mandatory-stmt]
                              [min-elements-stmt]
                              [max-elements-stmt]
                          "}") stmtsep

   not-supported-keyword-str = < a string that matches the rule >
                               < not-supported-keyword >

   add-keyword-str     = < a string that matches the rule >
                         < add-keyword >

   delete-keyword-str  = < a string that matches the rule >
                         < delete-keyword >

   replace-keyword-str = < a string that matches the rule >
                         < replace-keyword >

   ;; представляет использование расширения
   unknown-statement   = prefix ":" identifier [sep string] optsep
                         (";" /
                          "{" optsep
                              *((yang-stmt / unknown-statement) optsep)
                           "}") stmtsep

   yang-stmt           = action-stmt /
                         anydata-stmt /
                         anyxml-stmt /
                         argument-stmt /
                         augment-stmt /
                         base-stmt /
                         belongs-to-stmt /
                         bit-stmt /
                         case-stmt /
```

```
                        choice-stmt /
                        config-stmt /
                        contact-stmt /
                        container-stmt /
                        default-stmt /
                        description-stmt /
                        deviate-add-stmt /
                        deviate-delete-stmt /
                        deviate-not-supported-stmt /
                        deviate-replace-stmt /
                        deviation-stmt /
                        enum-stmt /
                        error-app-tag-stmt /
                        error-message-stmt /
                        extension-stmt /
                        feature-stmt /
                        fraction-digits-stmt /
                        grouping-stmt /
                        identity-stmt /
                        if-feature-stmt /
                        import-stmt /
                        include-stmt /
                        input-stmt /
                        key-stmt /
                        leaf-list-stmt /
                        leaf-stmt /
                        length-stmt /
                        list-stmt /
                        mandatory-stmt /
                        max-elements-stmt /
                        min-elements-stmt /
                        modifier-stmt /
                        module-stmt /
                        must-stmt /
                        namespace-stmt /
                        notification-stmt /
                        ordered-by-stmt /
                        organization-stmt /
                        output-stmt /
                        path-stmt /
                        pattern-stmt /
                        position-stmt /
                        prefix-stmt /
                        presence-stmt /
                        range-stmt /
                        reference-stmt /
                        refine-stmt /
                        require-instance-stmt /
                        revision-date-stmt /
                        revision-stmt /
                        rpc-stmt /
                        status-stmt /
                        submodule-stmt /
                        typedef-stmt /
                        type-stmt /
                        unique-stmt /
                        units-stmt /
                        uses-augment-stmt /
                        uses-stmt /
                        value-stmt /
                        when-stmt /
                        yang-version-stmt /
                        yin-element-stmt

   ;; Число элементов
   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

   ;; Даты
   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
                                 [1*key-predicate /
                                  leaf-list-predicate /
                                  pos]))

   key-predicate        = "[" *WSP key-predicate-expr *WSP "]"

   key-predicate-expr   = node-identifier *WSP "=" *WSP quoted-string

   leaf-list-predicate = "[" *WSP leaf-list-predicate-expr *WSP "]"

   leaf-list-predicate-expr = "." *WSP "=" *WSP quoted-string

   pos                  = "[" *WSP positive-integer-value *WSP "]"

   quoted-string        = (DQUOTE string DQUOTE) / (SQUOTE string SQUOTE)

   ;; 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, применяемые для регистро-чувствительных строк (RFC 7405)
   ;; Операторы
   action-keyword            = %s"action"
   anydata-keyword           = %s"anydata"
   anyxml-keyword            = %s"anyxml"
   argument-keyword          = %s"argument"
   augment-keyword           = %s"augment"
   base-keyword              = %s"base"
   belongs-to-keyword        = %s"belongs-to"
   bit-keyword               = %s"bit"
   case-keyword              = %s"case"
   choice-keyword            = %s"choice"
   config-keyword            = %s"config"
   contact-keyword           = %s"contact"
   container-keyword         = %s"container"
   default-keyword           = %s"default"
   description-keyword       = %s"description"
   deviate-keyword           = %s"deviate"
   deviation-keyword         = %s"deviation"
   enum-keyword              = %s"enum"
   error-app-tag-keyword     = %s"error-app-tag"
   error-message-keyword     = %s"error-message"
   extension-keyword         = %s"extension"
   feature-keyword           = %s"feature"
```

```
   fraction-digits-keyword  = %s"fraction-digits"
   grouping-keyword         = %s"grouping"
   identity-keyword         = %s"identity"
   if-feature-keyword       = %s"if-feature"
   import-keyword           = %s"import"
   include-keyword          = %s"include"
   input-keyword            = %s"input"
   key-keyword              = %s"key"
   leaf-keyword             = %s"leaf"
   leaf-list-keyword        = %s"leaf-list"
   length-keyword           = %s"length"
   list-keyword             = %s"list"
   mandatory-keyword        = %s"mandatory"
   max-elements-keyword     = %s"max-elements"
   min-elements-keyword     = %s"min-elements"
   modifier-keyword         = %s"modifier"
   module-keyword           = %s"module"
   must-keyword             = %s"must"
   namespace-keyword        = %s"namespace"
   notification-keyword     = %s"notification"
   ordered-by-keyword       = %s"ordered-by"
   organization-keyword     = %s"organization"
   output-keyword           = %s"output"
   path-keyword             = %s"path"
   pattern-keyword          = %s"pattern"
   position-keyword         = %s"position"
   prefix-keyword           = %s"prefix"
   presence-keyword         = %s"presence"
   range-keyword            = %s"range"
   reference-keyword        = %s"reference"
   refine-keyword           = %s"refine"
   require-instance-keyword = %s"require-instance"
   revision-keyword         = %s"revision"
   revision-date-keyword    = %s"revision-date"
   rpc-keyword              = %s"rpc"
   status-keyword           = %s"status"
   submodule-keyword        = %s"submodule"
   type-keyword             = %s"type"
   typedef-keyword          = %s"typedef"
   unique-keyword           = %s"unique"
   units-keyword            = %s"units"
   uses-keyword             = %s"uses"
   value-keyword            = %s"value"
   when-keyword             = %s"when"
   yang-version-keyword     = %s"yang-version"
   yin-element-keyword      = %s"yin-element"

   ;; Прочие ключевые слова
   add-keyword              = %s"add"
   current-keyword          = %s"current"
   delete-keyword           = %s"delete"
   deprecated-keyword       = %s"deprecated"
   false-keyword            = %s"false"
   invert-match-keyword     = %s"invert-match"
   max-keyword              = %s"max"
   min-keyword              = %s"min"
   not-supported-keyword    = %s"not-supported"
   obsolete-keyword         = %s"obsolete"
   replace-keyword          = %s"replace"
   system-keyword           = %s"system"
   true-keyword             = %s"true"
   unbounded-keyword        = %s"unbounded"
   user-keyword             = %s"user"
   and-keyword              = %s"and"
   or-keyword               = %s"or"
   not-keyword              = %s"not"

   current-function-invocation = current-keyword *WSP "(" *WSP ")"

   ;; 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

   identifier          = (ALPHA / "_")
                         *(ALPHA / DIGIT / "_" / "-" / ".")

   identifier-ref-arg-str = < a string that matches the rule >
```

```
                          < identifier-ref-arg >

identifier-ref-arg  = identifier-ref

identifier-ref      = [prefix ":"] identifier

string              = < an unquoted string, as returned by >
                      < the scanner, that matches the rule >
                      < yang-string >

yang-string         = *yang-char
;; любой символ Unicode или ISO/IEC 10646, включая табуляцию,
;; возврат каретки и перевод строки, но исключая другие управляющие
;; символы C0, суррогатные блоки и «несимволы»
yang-char = %x09 / %x0A / %x0D / %x20-D7FF /
                            ; исключить суррогатные блоки %xD800-DFFF
            %xE000-FDCF /   ; исключить несимвольные коды %xFDD0-FDEF
            %xFDF0-FFFD /   ; исключить несимвольные коды %xFFFE-FFFF
            %x10000-1FFFD / ; исключить несимвольные коды %x1FFFE-1FFFF
            %x20000-2FFFD / ; исключить несимвольные коды %x2FFFE-2FFFF
            %x30000-3FFFD / ; исключить несимвольные коды %x3FFFE-3FFFF
            %x40000-4FFFD / ; исключить несимвольные коды %x4FFFE-4FFFF
            %x50000-5FFFD / ; исключить несимвольные коды %x5FFFE-5FFFF
            %x60000-6FFFD / ; исключить несимвольные коды %x6FFFE-6FFFF
            %x70000-7FFFD / ; исключить несимвольные коды %x7FFFE-7FFFF
            %x80000-8FFFD / ; исключить несимвольные коды %x8FFFE-8FFFF
            %x90000-9FFFD / ; исключить несимвольные коды %x9FFFE-9FFFF
            %xA0000-AFFFD / ; исключить несимвольные коды %xAFFFE-AFFFF
            %xB0000-BFFFD / ; исключить несимвольные коды %xBFFFE-BFFFF
            %xC0000-CFFFD / ; исключить несимвольные коды %xCFFFE-CFFFF
            %xD0000-DFFFD / ; исключить несимвольные коды %xDFFFE-DFFFF
            %xE0000-EFFFD / ; исключить несимвольные коды %xEFFFE-EFFFF
            %xF0000-FFFFD / ; исключить несимвольные коды %xFFFFE-FFFFF
            %x100000-10FFFD ; исключить несимвольные коды %x10FFFE-10FFFF

integer-value       = ("-" non-negative-integer-value) /
                       non-negative-integer-value

non-negative-integer-value = "0" / positive-integer-value

positive-integer-value = (non-zero-digit *DIGIT)

zero-integer-value  = 1*DIGIT

stmtend             = optsep (";" / "{" stmtsep "}") stmtsep

sep                 = 1*(WSP / line-break)       ; 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 из 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

HTAB                = %x09 ; горизонтальная табуляция

LF                  = %x0A ; перевод строки

SP                  = %x20 ; пробел

WSP                 = SP / HTAB   ; пустое пространство (пробел)

<CODE ENDS>
```

15. Отклики NETCONF об ошибках, связанных с YANG

Множество откликов NETCONF 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.

15.1. Сообщение для данных, нарушающих unique

Если операция NETCONF будет давать конфигурацию, в которой ограничение unique перестаёт выполняться, **должна** возвращаться описанная ниже ошибка.

```
error-tag:      operation-failed
error-app-tag:  data-not-unique
error-info:     <non-unique>: содержит экземпляр идентификатора,
                указывающий на лист, нарушающий ограничение unique.
                Этот элемент указывает 1 раз для каждого
                неуникального leaf.

                Элемент <non-unique> относится к пространству имён
                YANG urn:ietf:params:xml:ns:yang:1.
```

15.2. Сообщение для данных, нарушающих max-elements

Если операция NETCONF будет создавать конфигурацию, где узел list или leaf-list включает слишком много элементов, **должна** возвращаться описанная ниже ошибка.

```
error-tag:      operation-failed
error-app-tag:  too-many-elements
```

Эта ошибка возвращается однократно и error-path указывает узел списка, даже при возникновении более одного избыточного потомка.

15.3. Сообщение для данных, нарушающих min-elements

Если операция NETCONF будет создавать конфигурацию, где узел list или leaf-list включает слишком мало элементов, **должна** возвращаться описанная ниже ошибка.

```
error-tag:      operation-failed
error-app-tag:  too-few-elements
```

Эта ошибка возвращается однократно и error-path указывает узел списка, даже при возникновении более одного недостающего потомка.

15.4. Сообщение для данных, нарушающих must

Если операция NETCONF будет создавать конфигурацию, где нарушены ограничения, заданные оператором must, **должна** возвращаться описанная ниже ошибка, если нет конкретного субоператора error-app-tag для оператора must.

```
error-tag:      operation-failed
error-app-tag:  must-violation
```

15.5. Сообщение для данных, нарушающих require-instance

Если операция NETCONF будет создавать конфигурацию, где лист типа instance-identifier или leafref с оператором require-instance, имеющим значение true, указывает на несуществующий экземпляр, **должна** возвращаться описанная ниже ошибка.

```
error-tag:      data-missing
error-app-tag:  instance-required
error-path:     путь к instance-identifier или leafref.
```

15.6. Сообщение для данных, нарушающих обязательный choice

Если операция NETCONF будет создавать конфигурацию, где не существует узлов в обязательном выборе (choice), **должна** возвращаться описанная ниже ошибка.

```
error-tag:      data-missing
error-app-tag:  missing-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.
```

15.7. 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, **должна** возвращаться описанная ниже ошибка.

```
error-tag:      bad-attribute
error-app-tag:  missing-instance
```

16. Взаимодействие с IANA

Этот документ регистрирует один идентификатор URN для возможности в реестре Network Configuration Protocol (NETCONF) Capability URNs.

Индекс	*Идентификатор возможности*
:yang-library	urn:ietf:params:netconf:capability:yang-library:1.0

17. 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 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 интерфейсов, применяемых при работе 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 с 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.

18. Литература

18.1. Нормативные документы

[ISO.10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[XML-NAMES] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

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

[XSD-TYPES] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

18.2. Дополнительная литература

[CoMI] van der Stok, P. and A. Bierman, "CoAP Management Interface", Work in Progress, draft-vanderstok-core-comi-09, March 2016.

[IEEE754-2008] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2008, DOI 10.1109/IEEESTD.2008.4610935, 2008, <http://standards.ieee.org/findstds/standard/754-2008.html>.

[RESTCONF] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", Work in Progress[1], draft-ietf-netconf-restconf-16, August 2016.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, <http://www.rfc-editor.org/info/rfc2579>.

[1]Работа опубликована в RFC 8040. *Прим. перев.*

[RFC3780] Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation Structure of Management Information", RFC 3780, DOI 10.17487/RFC3780, May 2004, <http://www.rfc-editor.org/info/rfc3780>.

[RFC4844] Daigle, L., Ed., and Internet Architecture Board, "The RFC Series and RFC Editor", RFC 4844, DOI 10.17487/RFC4844, July 2007, <http://www.rfc-editor.org/info/rfc4844>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <http://www.rfc-editor.org/info/rfc6643>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

[XPATH2.0] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, December 2010, <http://www.w3.org/TR/2010/REC-xpath20-20101214>.

[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>.

[YANG-Guidelines] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", Work in Progress[1], draft-ietf-netmod-rfc6087bis-07, July 2016.

Благодарности

Редактор благодарит перечисленных ниже людей, которые предоставили полезные комментарии к предварительным вариантам этого документа: Mehmet Ersue, Washam Fan, Joel Halpern, Per Hedeland, Leif Johansson, Ladislav Lhotka, Lionel Morand, Gerhard Muenz, Peyman Owladi, Tom Petch, Randy Presuhn, David Reid, Jernej Tuljak, Kent Watsen, Bert Wijnen, Robert Wilton и Dale Worley.

Участники работы

Ниже перечислены люди, внёсшие существенный вклад в разработку исходного документа YANG:

- Andy Bierman (YumaWorks)
- Balazs Lengyel (Ericsson)
- David Partain (Ericsson)
- Juergen Schoenwaelder (Jacobs University Bremen)
- Phil Shafer (Juniper Networks)

Адрес автора

Martin Bjorklund (редактор)

Tail-f Systems

Email: mbj@tail-f.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru

[1]Работа опубликована в RFC 8407. *Прим. перев.*