Internet Engineering Task Force (IETF)
Request for Comments: 8342
Updates: 7950
Category: Standards Track
ISSN: 2070-1721

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems

March 2018

# Архитектура хранилища данных сетевого управления NMDA

# Network Management Datastore Architecture (NMDA)

**Тезисы**

Хранилища данных являются фундаментальной концепцией привязки моделей данных, написанных на языке моделирования YANG, к протоколам сетевого управления типа NETCONF[1] и RESTCONF. Этот документ определяет архитектурную модель для хранилищ данных на основе опыта, полученного с начальными простыми моделями, которая решает вопросы, не поддерживаемые первоначальной моделью. Документ обновляет RFC 7950.

**Статус документа**

Документ относится к категории Internet Standards Track.

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

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc8342.



# Оглавление

1. Введение....2
2. Цели....2
3. Терминология....2
4. Предпосылки....4
4.1. Исходная модель хранилищ данных....4
5. Архитектурная модель хранилищ данных....5
5.1. Традиционные хранилища конфигурации....5
5.1.1. Хранилище стартовой конфигурации (<startup>)....5
5.1.2. Хранилище будущей конфигурации (<candidate>)....6
5.1.3. Хранилище рабочей конфигурации (<running>)....6
5.1.4. Хранилище предполагаемой конфигурации (<intended>)....6
5.2. Хранилища динамической конфигурации....6
5.3. Хранилище операционного состояния (<operational>)....6
5.3.1. Остаточная конфигурация....7
5.3.2. Отсутствующие ресурсы....7
5.3.3. Контролируемые системой ресурсы....7
5.3.4. Аннотация метаданных источника....7
6. Воздействие на YANG....8
6.1. Контекст XPath....8
6.2. Вызовы операций и RPC....8
7. Модули YANG....8
8. Взаимодействие с IANA....12
8.1. Обновление реестра IETF XML Registry....12
8.2. Обновление реестра YANG Module Names Registry....12
9. Вопросы безопасности....12
10. Литература....12

[1]Network Configuration Protocol — протокол настройки сети.
[2]Internet Engineering Task Force.
[3]Internet Engineering Steering Group.

10.1. Нормативные документы....12
10.2. Дополнительная литература....12
Приложение A. Рекомендации по определению хранилищ....13
A.1. Определение модулей YANG, применимых для хранилища....13
A.2. Определение применимого подмножества операторов YANG....13
A.3. Определение способов актуализации данных....13
A.4. Определение применимых протоколов....13
A.5. Определение отождествлений YANG для хранилища....13
Приложение B. Пример эфемерного хранилища....13
Приложение C. Примеры....14
C.1. Пример хранилища System....14
C.2. Пример BGP....16
C.2.1. Хранилища данных....16
C.2.2. Добавление партнера....16
C.2.2.1. Хранилище <operational>....17
C.2.3. Удаление партнера....17
C.3. Пример интерфейса....17
C.3.1. Заранее представленные интерфейсы....18
C.3.2. Предоставляемые системой интерфейсы....18
Благодарности....18
Адреса авторов....19

# 1. Введение

Этот документ предоставляет архитектурную модель для хранилищ данных, используемых протоколами сетевого управления типа NETCONF [RFC6241] и RESTCONF [RFC8040], а также языком моделирования данных YANG [RFC7950]. Хранилища являются фундаментальной концепцией, привязывающей модели данных сетевого управления к протоколам сетевого управления. Согласование общей архитектурной модели хранилища данных позволит создавать модели данных без привязки к конкретному протоколу сетевого управления. Эта архитектурная основа идентифицирует набор концептуальных хранилищ, но не обязывает все протоколы сетевого управления предоставлять все эти концептуальные хранилища. Архитектура не привязана к кодированию, используемому протоколами сетевого управления.

Данный документ обновляет RFC 7950 в части определения доступного дерева для некоторых вариантов контекста XPath[1] (см. параграф 6.1) и контекста вызова операций ( см. параграф 6.2).

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

# 2. Цели

Объекты данных сетевого управления зачастую могут иметь два значения — одно значение задано пользователем или приложением (конфигурация), а второе используется устройством в настоящий момент (текущее состояние). Эти два значения по многим причинам (например, внутреннее взаимодействие системы с оборудованием, взаимодействие с протоколами и другими устройствами или просто временной интервал на передачу изменений конфигурации в программные и аппаратные компоненты системы) могут различаться. Кроме того, конфигурационные данные и данные текущего состояния могут различаться по срокам действия.

Исходная модель хранилищ данных требует моделирования этих данных в схеме YANG дважды — как объекты config true и config false. Соглашение, принятое в модели данных интерфейса [RFC8343] и модели данных IP [RFC8344], состоит в использовании двух раздельных ветвей с общим корнем — одна ветвь для объектов данных конфигурации, другая для объектов данных текущего состояния.

Дублирование определений и специальное разделение данных текущего состояния и конфигурации порождало множество проблем. Размещение данных конфигурации и состояния в разных ветвях модели данных усложняет работу с моделями и снижает уровень их читаемости. Кроме того, связи между ветвями не читаемы для машины, а выражения фильтров работающих с данными конфигурации и связанными с ними данными состояния отличаются.

В пересмотренной архитектурной модели хранилищ данных объекты определяются в схеме YANG только один раз, но их независимые экземпляры могут присутствовать в разных хранилищах данных, например, один для настраиваемого значения, а другой для текущего. Это обеспечивает более простое и изящное решение проблемы.

Пересмотренная архитектурная модель хранилищ данных поддерживает дополнительные хранилища для систем, использующих более изощренную обработку преобразования конфигурационных данных в рабочее состояние. Например, некоторые системы могут поддерживать конфигурацию, которая в настоящее время не применяется (неактивная конфигурация), или конфигурационные шаблоны, которые служат для преобразования данных конфигурации.

# 3. Терминология

Используемые в документе термины определены ниже. Некоторые термины используют пересмотренные определения из [RFC6241] и [RFC7950] (см. также раздел 4). Эти пересмотренные определения семантически эквивалентны определениям из [RFC6241] и [RFC7950]. Предполагается, что обновленные определения из этого раздела будут применяться взамен определений [RFC6241] и [RFC7950] при пересмотре этих документов.

***datastore – хранилище данных***

Концептуальное место для хранения и доступа к информации. Хранилище может быть реализовано, например, в виде файла, базы данных, flash-памяти или их комбинации. Хранилище отображается на экземпляр дерева данных YANG.

***schema node - узел схемы***

[1]XML Path Language — язык путей XML.

Узел в дереве схемы. Формальное определение приведено в RFC 7950.

***datastore schema - схема хранилища данны***

Комбинированный набор узлов схемы для всех модулей, поддерживаемых конкретным хранилищем с учетом всех отклонений и активных функций (feature) хранилища.

***configuration – конфигурация, данные конфигурации***

Данные, которые нужны для перевода устройства из начального состояния в желаемое рабочее состояние. Эти данные моделируются в YANG с использованием узлов config true. Конфигурация может происходить из разных источников.

***configuration datastore – хранилище конфигурации***

Хранилище данных, содержащее параметры конфигурации.

***running configuration datastore – хранилище рабочей конфигурации***

Конфигурационное хранилище, содержащее текущие (рабочие) параметры конфигурации устройства. Оно может включать конфигурацию, которая перед применением требует тех или иных преобразований. Это хранилище обозначают <running>.

***candidate configuration datastore – хранилище будущей конфигурации, хранилище-кандидат***

Конфигурационное хранилище, данные в котором можно изменять без влияния на рабочее хранилище данных устройства. Впоследствии эти данные могут быть переданы в рабочее хранилище. Это хранилище обозначают <candidate>.

***startup configuration datastore - хранилище стартовой конфигурации***

Конфигурационное хранилище, содержащие данные, которые будут помещены в рабочее хранилище при следующей загрузке устройства. Это хранилище обозначают <startup>.

***intended configuration – предполагаемая конфигурация***

Конфигурация, предназначенная для использования устройством. Представляет собой конфигурацию после выполнения всех преобразований в <running> и является конфигурацией, которую система попытается применить.

***intended configuration datastore – хранилище предполагаемой конфигурации***

Конфигурационное хранилище содержащее полную предполагаемую конфигурацию. Это хранилище обозначают <intended>.

***configuration transformation - преобразование конфигурации***

Дополнение, изменение или удаление данных конфигурации при передаче между хранилищами <running> и <intended>. Примерами конфигурационных преобразований служат удаление из конфигурации неактивных элементов и создание конфигурации из шаблона.

***conventional configuration datastore – традиционное (обычное) хранилище данных конфигурации***

Одно из хранилищ <running>, <startup>, <candidate> и <intended>. Эти хранилища используют общую схему и протокольные операции позволяют копировать данные между хранилищами. Термин conventional выбран в качестве общего обозначения всех таких хранилищ.

***conventional configuration – конфигурация общего назначения***

Конфигурационные данные, хранящиеся в любом из традиционных хранилищ.

***dynamic configuration datastore – хранилище динамических данных конфигурации***

Конфигурационное хранилище, содержащее данные, полученные динамически в процессе работы устройства путем взаимодействия с другими системами, а не из традиционного хранилища.

***dynamic configuration – динамическая конфигурация***

Конфигурация, полученная из динамического хранилища.

***learned configuration — выведенная конфигурация***

Конфигурация, которая была выведена (learned) путем протокольного взаимодействия с другими системами и не является ни традиционной, ни динамической.

***system configuration – системная конфигурация***

Конфигурация, поставляемая (обеспечиваемая) самой системой.

***default configuration – принятая по умолчанию конфигурация***

Конфигурация, которая не задается явно, а использует принятые по умолчанию значения используемой модели данных.

***applied configuration – примененная конфигурация***

Конфигурация, которая используется в настоящий момент устройством. Примененная конфигурация происходит от традиционной, динамической, выведенной или принятой по умолчанию конфигурации.

***system state – состояние системы***

Дополнительные данные системы, которые не относятся к конфигурационным, это могут быть предназначенные только для чтения параметры состояния или собранная статистика. Состояние системы является временным и меняется в результате взаимодействия с внутренними компонентами или внешними системами. Состояние системы моделируется в YANG с использованием узлов config false.

***operational state – рабочее состояние***

Комбинация примененной конфигурации и состояния системы.

***operational state datastore – хранилище рабочего состояния***

Хранилище данные, содержащее полное рабочее состояние устройства. Это хранилище обозначают <operational>.

***origin – источник, происхождение***

Аннотация метаданных, показывающая источник элементе данных.

***remnant configuration – остаточная конфигурация***

Конфигурация, которая остается частью примененной конфигурации в течение некоторого времени после ее удаления из предполагаемой или динамической конфигурации. Интервал сохранения может быть минимальным или длиться до тех пор, пока не будут освобождены все использованные удаленной конфигурацией ресурсы (например, сетевые соединения, память, идентификаторы файлов).

Приведенные ниже термины не связаны напрямую с хранилищами данных, но часто используются в документе.

***client - клиент***

Объект, который может иметь доступ к данным YANG на сервере с помощью некого протокола управления сетью.

***server - сервер***

Объект, который пердоставляет клиенту доступ к данным YANG с помощью некого протокола управления сетью.

***notification - уведомление***

Инициированное сервером сообщение, указывающее на некое событие, которое распознал сервер.

***remote procedure call – вызов удаленной процедуры***

Операция которая может быть вызвана клиентом для выполнения на сервере.

# *4. Предпосылки*

Протокол NETCONF [RFC6241] включает приведенные ниже определения.

***datastore – хранилище данных***

Концептуальное место хранения информации и доступа к ней. Хранилище может быть реализовано с использованием файлов, баз данных, флэш-памяти или их комбинации.

***configuration datastore - хранилище конфигурации***

Хранилище данных, содержащее полный набор параметров конфигурации, требуемых для перевода устройства из начального состояния в желаемое рабочее состояние.

Язык YANG 1.1 [RFC7950] вносит уточнения для использования NETCONF с YANG (обычное дело, но отметим, что протокол NETCONF был создан раньше YANG).

***datastore***

При моделировании с использованием YANG хранилеще данных реализуется в виде экземпляра дерева данных.

***configuration datastore***

При моделировании с использованием YANG хранилеще данных реализуется в виде экземпляра дерева конфигурационных данных.

[RFC6244] определяет данные операционного состояния, как показано ниже.

- Данные операционного состояния представляют собой набор данных, получаемых системой во время работы и влияющих на поведение системы, подобно конфигурационным данным. В отличие от данных конфигурации операционное состояние является временным и меняется в результате взаимодействий с внутренними компонентами или другими системами по специальным протоколам.

В параграфе 4.3.3 [RFC6244] рассматривается операционное состояние и среди прочего упоминается возможность рассматривать это состояние, как сохраняемое в другом хранилище данных. В параграфе 4.4 [RFC6244] делается заключение, что во время написания документа рекомендовалось моделирование состояния в виде отдельных листьев и ветвей.

Опыт реализации и запросы операторов [OpState-Reqs] [OpState-Modeling] показали, что модель хранилища, разработанная изначально для NETCONF и уточненная YANG, требует расширения. В частности, были разработаны понятия предполагаемой конфигурации и примененной конфигурации.

## 4.1. Исходная модель хранилищ данных

На рисунке 1 показана исходная модель хранилищ данных, используемая сейчас протоколом NETCONF [RFC6241].

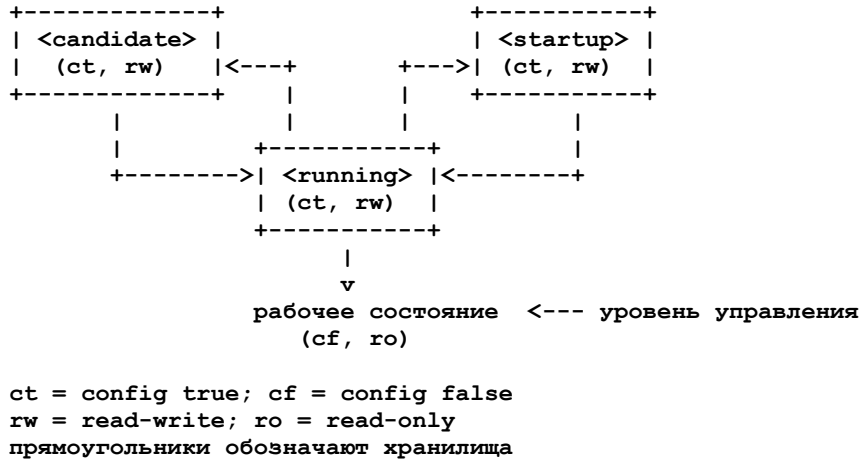


*Рисунок 1.*

Отметим, что модель на рисунке упрощена, полномочия read-only (ro) и read-write (rw) представлены с точки зрения клиента на концептуальном уровне. В NETCONF, например, поддержка хранилищ <candidate> и <startup> является не обязательной, а запись в хранилище <running> не разрешена. Кроме того, хранилище <startup> может быть изменено только путем копирования в него хранилища <running> <startup> в стандартизованной модели редактирования хранилищ NETCONF. Протокол RESTCONF не показывает таких различий и вместо этого обеспечивает универсальное хранилище с возможностью записи, которое скрывает редактирование через промежуточное хранилище <candidate> путем прямой записи в <running> или иного механизма, зависящего от реализации. RESTCONF также скрывает, как конфигурация становится постоянной. Отметим, что реализации могут иметь дополнительные хранилища, которые могут распространять изменения на <running>. NETCONF явно указывает «именованные хранилища данных» (named datastore).

Ниже приведены некоторые соображения по результатам использования исходной модели.

- Операционное состояние не было определено как хранилище данных, хотя были предложения о создании такого хранилища.
- Операция NETCONF <get> возвращает содержимое хранилища <running> вместе с операционным состоянием. Это требует хранения данных config false и config true вразных ветвях, если данные операционного состояния по сроку жизни отличаются от конфигурационных данных, в также в тех случаях, когда конфигурация применяется не сразу или возникают проблемы с ее применением.
- Некоторые реализации используют фирменные механизмы, которые позволяют клиентам сохранять неактивные данные в хранилище <running>. Концептуально такие данные удаляются перед проверкой пригодности конфигурации.

- Некоторые реализации используют фирменные механизмы, которые позволяют клиентам определять шаблоны конфигурации в хранилище <running>. Эти шаблоны автоматически раскрываются (преобразуются) системой и полученная в результате конфигурация применяется внутренними средствами.
- Некоторые операторы отмечали, что для них важно получить успешно примененую конфигурацию, которая может быть надмножеством или подможеством конфигурации в хранилище <running>.

# *5. Архитектурная модель хранилищ данных*

На рисунке 2 представлена новая концептуальная модель хранилищ данных, расширяющая исходную модель в соответствии с набранным при ее использовании опытом.


*Рисунок 2.*

## 5.1. Традиционные хранилища конфигурации

Традиционные хранилища конфигурации представляют собой набор хранилищ данных, которые используют общую схему, что позволяет копировать данные из одного хранилища в другое. Термин используется для единого обозначения всех традиционных хранилищ. Если модуль не содержит конфигурационных данных и не требуется для импорта в другие модели, иго **можно** не включать в схему для традиционных хранилищ данных. Набор традиционных хранилищ включает:

- <running>
- <candidate>
- <startup>
- <intended>

В будущих документах могут быть определены дополнительные хранилища.

Поток данных между хранилищами описан в разделе 5.

Конкретные протоколы могут определять явные операции копирования между хранилищами (например, NETCONF <copy-config>.

### *5.1.1. Хранилище стартовой конфигурации (<startup>)*

Хранилище стартовой конфигурации <startup> содержит данные, которые устройство использует при своей загрузке (включении). Хранилище <startup> присутствует только на устройствах, имеющих раздельные хранилища для стартовой и рабочей конфигурации.

Хранилище стартовой конфигурации может поддерживаться не всеми протоколами и реализациями.

На устройствах с энерго-независимой памятью содержимое хранилища <startup> обычно будет сохраняться при перезагрузке устройства с использованием этого хранилища. Во время загрузки устройство помещает стартовую конфигурацию в хранилище <running>. Для сохранения новой стартовой конфигурации данные копируются в <startup> с помощью явной или неявной операции протокола.

### 5.1.2. Хранилище будущей конфигурации (<candidate>)

Хранилище будущей конфигурации <candidate> представляет собой хранилище конфигурационных данных, которыми можно манипулировать без влияния на текущую конфигурацию устройства, а затем представлять в хранилище <running>.

Хранилище будущей конфигурации может поддерживаться не всеми протоколами и реализациями.

Хранилище <candidate> обычно не сохраняется при перезагрузке устройства, даже если для него используется энерго-независимая память. Если хранилище <candidate> сохраняется в такой памяти, в процессе перезагрузки оно заменяется содержимым хранилища <running>.

### 5.1.3. Хранилище рабочей конфигурации (<running>)

Хранилище рабочей конфигурации <running> представляет собой хранилище с текущей конфигурацией устройства. Оно **может** содержать конфигурацию, которая перед применением требует преобразования (например, удаление неактивных элементов или раскрытие шаблонов). Однако хранилище <running> всегда **должно** быть пригодным деревом конфигурационных данных, как указано в параграфе 8.1 [RFC7950].

Хранилище <running> **должно** поддерживаться, если устройство настраивается с использованием традиционного хранилища конфигурации.

Если устройство не имеет отдельно хранилища <startup> и доступна энерго-независимая память, такое устройство обычно будет использовать эту память для сохранения <running> в процессе перезагрузки.

### 5.1.4. Хранилище предполагаемой конфигурации (<intended>)

Хранилище предполагаемой конфигурации <intended> открыто только для чтения. Оно представляет конфигурацию после выполнения всех преобразования в <running> (например, расширения шаблонов и удаления неактивных компонент), которую система пытается применить.

Хранилище <intended> тесно связано с <running>. При записи данных в хранилище <running> сервер **должен** незамедлительно обновить и проверить на пригодность хранилище <intended>.

Содержимое <intended> **может** также обновляться независимо от <running>, если воздействие преобразований конфигурации меняется, но хранилище <intended> всегда **должен** быть пригодным деревом данных конфигурации, как указано в параграфе 8.1 [RFC7950].

В простых реализациях хранилища <running> и <intended> идентичны.

Содержимое <intended> также связано с подмножеством config true хранилища <operational>, поэтому клиент может определить, в какой степени предполагаемая конфигурация соответствует текущей, просто проверяя, какая часть содержимого <intended> присутствует в <operational>.

Хранилище <intended> не сохраняется при перезагрузке — его привязка к <running> делает это ненужным.

В настоящее время не определено стандартных механизмов воздействия на хранилище <intended> так, чтобы оно отличалось от <running>, но данная архитектура позволяет определить такие механизмы.

Примером такого механизма может служить поддержка маркировки неактивных узлов в хранилище <running>. Неактивные узлы не будут копироваться в хранилище <intended>. Вторым примером является поддержка шаблонов, которые могут выполнять преобразования конфигурации из <running> для записи в хранилище <intended>.

## 5.2. Хранилища динамической конфигурации

Модель принимает необходимость хранилищ динамической конфигурации, которые по определению не являются частью постоянной конфигурации устройства. Иногда такие хранилища называют эфемерными (ephemeral datastore), поскольку хранящаяся в них информация теряется при перезагрузке. Хранилища динамической конфигурации взаимодействуют с остальной системой через хранилище <operational>.

Схема для хранилища динамических данных **может** отличаться от схем традиционных хранилищ. Если модуль не содержит узлов данных конфигурации и не требуется для импорта в другие модули, он **может** не включаться в схему хранилища динамической конфигурации.

## 5.3. Хранилище операционного состояния (<operational>)

Хранилище операционного состояния (<operational>) открыто лишь для чтения и включает все узлы config true и config false, определенные в схеме хранилища данных. В исходной модели NETCONF модель операционного состояния содержит только узлы config false. Причиной включения узлов config true является потребность в представлении рабочих параметров без их дублирования в модели данных.

Хранилище <operational> содержит данные состояния и конфигурацию, реально используемую системой. Это включает примененную конфигурацию из хранилища <intended>, выведенную (learned) и представленную системой конфигурацию, а также значения, принятые по умолчанию, которые определены любыми поддерживаемыми моделями данных. В дополнение к этому <operational> содержит также примененную конфигурацию из хранилища динамической конфигурации.

Схема для хранилища <operational> **должна** быть надмножеством комбинированной схемы, используемой во всех конфигурационных хранилищах, однако узлы конфигурационных данных **могут** не включаться в хранилище <operational>, если сервер не способен точно сообщать о них.

Запросы на поиск узлов из хранилища <operational> всегда возвращают используемое значение для существующих узлов, независимо от наличия в модуле YANG принятых по умолчанию значений. Если для данного узла значение не возвращено, это означает, что узел не используется устройством.

Интерпретация слов «используется системой» (in use) зависит от определения схемы и реализации устройства. Обычно функциональность, которая разрешена (включена) и работает в системе, считается используемой. И наоборот, функциональность, которая не включена и не работает считается «не используемой», поэтому ее не **следует** включать в <operational>.

Для хранилища <operational> **следует** соблюдать любые ограничения, указанные в модели данных, но с учетом главной цели возвращать «используемые» значения, возможны случаи, когда такие ограничения **могут** быть нарушены при некоторых обстоятельствах (например, использование аномального значения, изменении структуры списка или

наличии остаточной конфигурации (см. параграф 5.3.1)). Отметим, что такие отклонения **следует** допускать лишь в тех случаях, когда заранее известно, что устройство не полностью соответствует схеме <operational>.

Нарушаться **могут** только семантические ограничения. К ним относятся операторы YANG when, must, mandatory, unique, min-elements и max-elements, а также уникальность значений ключей.

Синтаксические ограничения, включая иерархическую организацию, идентификаторы и ограничения по типам, нарушать **недопустимо**. Если узел в хранилище <operational> не соответствует синтаксическим ограничениям, **недопустимо** возвращать его, а для информирования об ошибке следует применять другие механизмы.

Хранилище <operational> не сохраняется при перезагрузке устройства.

### *5.3.1. Остаточная конфигурация*

При изменении конфигурации может потребоваться некоторое время на прохождение через хранилище <operational>. В этом интервале хранилище <operational> может одновременно содержать узлы предшествующей и текущей конфигурации, насколько возможно точно отслеживая текущую работу устройства. Остаточные данные предыдущей конфигурации будут сохраняться до тех пор, пока система не освободит ресурсы, использованные недавно удаленной конфигурацией (например, сетевые соединения, память, файлы).

Остаточная конфигурация является типичным примером, где семантические ограничения, определенные в модели данных, не могут быть применены для хранилища <operational>, поскольку в системе может сохраняться конфигурация, ограничения которой были пригодны для прежнего варианта, но недействительны в текущей конфигурации. Поскольку ограничения узлов config false могут быть связаны с узлами config true, остаточная конфигурация может вызывать нарушение этих ограничений.

### *5.3.2. Отсутствующие ресурсы*

Конфигурация в <intended> может указывать ресурсы, которые не доступны или физически отсутствуют. В такой ситуации соответствующие части <intended> не применяются. Данные этих разделов присутствуют в <intended>, но не появляются в <operational>.

Типичным примером является конфигурация интерфейса, который в настоящее не присутствует. В этом случае конфигурация интерфейса остается в хранилище <intended>, но не включается в <operational>.

Отметим, что пригодность конфигурации не может зависеть от текущего состояния таких ресурсов, поскольку это приводило бы в непригодности конфигурации в случае удаления ресурса. Это неприемлемо, особенно с учетом того, что перезагрузка такого устройства будет приводить к его перезапуску с непригодной конфигурацией. Поэтому конфигурации с отсутствующими ресурсами разрешаются для хранилищ <running> и <intended>, но эти ресурсы не могут присутствовать в <operational>.

### *5.3.3. Контролируемые системой ресурсы*

Иногда ресурсы, контролируемые системой и соответствующие данные появляются (и исчезают) в хранилище <operational> динамически. Если контролируемый системой ресурс имеет соответствующую конфигурацию в <intended> при его появлении, система попытается применить эту конфигурацию и это в конечном итоге приведет к ее появлению в хранилище <operational> (если применение пройдет успешно).

### *5.3.4. Аннотация метаданных источника*

При передаче конфигурации в хранилище <operational> она концептуально помечается аннотацией метаданных [RFC7952]Ю указывающей источник. Источник применяет все узлы за исключением контейнеров отсутствия. Аннотация метаданных origin определена в разделе 7. Значениями служат отождествления YANG, перечисленные ниже.

- **origin** - абстрактное базовое отождествление, из которого выводено другое отождествление источника.
- **intended** - представляет конфигурацию из хранилища <intended>.
- **dynamic** - представляет конфигурацию из хранилища динамической конфигурации.
- **system** - представляет конфигурацию, обеспеченную самой системой. Примеры системной конфигурации включают всегда присутствующий интерфейс loopback или конфигурацию интерфейса, которая создается автоматически при наличии оборудования в устройстве.
- **learned** - представляет конфигурацию, которая была выведена через протокольные взаимодействия с другими системами (включая такие взаимодействия как согласование канального уровня, протоколы маршрутизации, DHCP).
- **default** - представляет конфигурацию с применением значений, принятых по умолчанию в модели данных, используя значения в операторе default или любые значения, описанные в операторе description. Этот источник лишь про отсутствии других источников конфигурации.
- **unknown** - представляет конфигурацию, для которой система не может определить источник.

Эти отождествления в дальнейшем могут быть уточнены, например, могут быть разделены отождествления отдельных типов или экземпляров хранилищ динамической конфигурации, созданных на основе dynamic.

Для всех узлов конфигурационных данных в хранилище <operational> устройству **следует** указывать источник, наиболее точно соответствующих происхождению конфигурации, используемой системой.

При возникновении неоднозначностей в части выбора источника, когда конфигурационные данные приходят из нескольких источников сразу, **следует** использовать оператор description в модуле YANG для выбора подходящего источника. Например,

> Если для отдельного узла конфигурации соответствующий оператор description в модуле YANG указывает, что согласованное протоколом значение переписыват значение из конфигурации, в качестве источника следует указывать learned, даже при совпадении выведенного значения с указанным в конфигурации.
>
> И наоборот, если для отдельного узла конфигурации соответствующий оператор description в модуле YANG указывает, что согласованное протоколом значение не переписыват значение из конфигурации, в качестве источника следует указывать intended, даже при совпадении выведенного значения с заданным в конфигурации.

Если устройство не может точно указать источник данных для отдельного узла конфигурации, **следует** указывать источник unknown.

# 6. Воздействие на YANG

## 6.1. Контекст XPath

Этот параграф обновляет параграф 6.4.1 из RFC 7950.

Если сервер реализует определенную в этом документе архитектуру, доступные деревья для некоторых вариантов контекста XPath уточняются как показано ниже.

- Если выражение XPath определено в субоператоре узла данных, который представляет состояние системы, доступным деревом является все операционное состояние сервера. Корневой узел имеет в качестве потомков узлы верхнего уровня всех модулей.
- Если выражение XPath определено в субоператоре оператора notification, доступным деревом является экземпляр уведомления и все операционное состояние сервера. Если уведомление определено на вершине модуля, корневой узел имеет в качестве потомков узлел, представляющий уведомление, которое определено, и узлы верхнего уровня всех модулей. В остальных случаях корневой узел имеет в качестве потомков узлы верхнего уровня всех модулей.
- Если выражение XPath определено в субоператоре оператора input внутри оператора rpc или action, доступным деревом будет экземпляр RPC или операции и все операционное состояние сервера. Корневой узел имеет в качестве потомков узлы верхнего уровня всех модулей. Кроме того, для RPC корневой узел имеет в качестве потомка также узел, который представляет определяемую операцию RPC. Этот узел имеет в качестве потомков входные параметры операции.
- Если выражение XPath определено в субоператоре оператора output внутри оператора rpc или action, доступным деревом будет экземпляр RPC или операции и все операционное состояние сервера. Корневой узел имеет в качестве потомков узлы верхнего уровня всех модулей. Кроме того, для RPC корневой узел имеет в качестве потомка также узел, который представляет определяемую операцию RPC. Этот узел имеет в качестве потомков выходные результаты операции.

## 6.2. Вызовы операций и RPC

Этот параграф обновляет параграф 7.15 из RFC 7950.

Операции всегда вызываются к контексте хранилища операционного состояния. Узел, для которого вызывается операция, **должен** существовать в хранилище операционного состояния.

Отметим, что этот документ никак не ограничивает результат вызова RPC или операции. Например, можно определить RPC для изменения содержимого того или иного хранилища данных.

# 7. Модули YANG

```
<CODE BEGINS> file "ietf-datastores@2018-02-14.yang"

module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

  organization
    "IETF Network Modeling (NETMOD) Working Group";

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

     WG List:  <mailto:netmod@ietf.org>

     Author:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

     Author:   Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>

     Author:   Phil Shafer
               <mailto:phil@juniper.net>

     Author:   Kent Watsen
               <mailto:kwatsen@juniper.net>

     Author:   Rob Wilton
               <rwilton@cisco.com>";

  description
    "This YANG module defines a set of identities for identifying
     datastores.

     Copyright (c) 2018 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
```

```
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC 8342
       (https://www.rfc-editor.org/info/rfc8342); see the RFC itself
       for full legal notices.";

     revision 2018-02-14 {
       description
         "Initial revision.";
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }

     /*
      * Отождествления
      */

     identity datastore {
       description
         "Abstract base identity for datastore identities.";
     }

     identity conventional {
       base datastore;
       description
         "Abstract base identity for conventional configuration
          datastores.";
     }

     identity running {
       base conventional;
       description
         "The running configuration datastore.";
     }

     identity candidate {
       base conventional;
       description
         "The candidate configuration datastore.";
     }

     identity startup {
       base conventional;
       description
         "The startup configuration datastore.";
     }

     identity intended {
       base conventional;
       description
         "The intended configuration datastore.";
     }

     identity dynamic {
       base datastore;
       description
         "Abstract base identity for dynamic configuration datastores.";
     }

     identity operational {
       base datastore;
       description
         "The operational state datastore.";
     }

     /*
      * Определения типов
      */

     typedef datastore-ref {
       type identityref {
         base datastore;
       }
       description
```

```
        "A datastore identity reference.";
    }
  }

  <CODE ENDS>
```

```
  <CODE BEGINS> file "ietf-origin@2018-02-14.yang"

  module ietf-origin {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-origin";
    prefix or;

    import ietf-yang-metadata {
      prefix md;
    }

    organization
      "IETF Network Modeling (NETMOD) Working Group";

    contact
      "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

       WG List:  <mailto:netmod@ietf.org>

       Author:   Martin Bjorklund
                 <mailto:mbj@tail-f.com>

       Author:   Juergen Schoenwaelder
                 <mailto:j.schoenwaelder@jacobs-university.de>

       Author:   Phil Shafer
                 <mailto:phil@juniper.net>

       Author:   Kent Watsen
                 <mailto:kwatsen@juniper.net>

       Author:   Rob Wilton
                 <rwilton@cisco.com>";

    description
      "This YANG module defines an 'origin' metadata annotation and a
       set of identities for the origin value.

       Copyright (c) 2018 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Simplified BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC 8342
       (https://www.rfc-editor.org/info/rfc8342); see the RFC itself
       for full legal notices.";

    revision 2018-02-14 {
      description
        "Initial revision.";
      reference
        "RFC 8342: Network Management Datastore Architecture (NMDA)";
    }

    /*
     * Отождествления
     */

    identity origin {
      description
        "Abstract base identity for the origin annotation.";
    }

    identity intended {
```

```
    base origin;
    description
      "Denotes configuration from the intended configuration
       datastore.";
  }

  identity dynamic {
    base origin;
    description
      "Denotes configuration from a dynamic configuration
       datastore.";
  }

  identity system {
    base origin;
    description
      "Denotes configuration originated by the system itself.

       Examples of system configuration include applied configuration
       for an always-existing loopback interface, or interface
       configuration that is auto-created due to the hardware
       currently present in the device.";
  }

  identity learned {
    base origin;
    description
      "Denotes configuration learned from protocol interactions with
       other devices, instead of via either the intended
       configuration datastore or any dynamic configuration
       datastore.

       Examples of protocols that provide learned configuration
       include link-layer negotiations, routing protocols, and
       DHCP.";
  }

  identity default {
    base origin;
    description
      "Denotes configuration that does not have a configured or
       learned value but has a default value in use.  Covers both
       values defined in a 'default' statement and values defined
       via an explanation in a 'description' statement.";
  }

  identity unknown {
    base origin;
    description
      "Denotes configuration for which the system cannot identify the
       origin.";
  }

  /*
   * Определения типов
   */

  typedef origin-ref {
    type identityref {
      base origin;
    }
    description
      "An origin identity reference.";
  }

  /*
   * Metadata annotations
   */

  md:annotation origin {
    type origin-ref;
    description
      "The 'origin' annotation can be present on any configuration
       data node in the operational state datastore.  It specifies
       from where the node originated.  If not specified for a given
       configuration data node, then the origin is the same as the
```

```
          origin of its parent node in the data tree.  The origin for
          any top-level configuration data nodes must be specified.";
      }
    }

    <CODE ENDS>
```

# 8. Взаимодействие с IANA

## 8.1. Обновление реестра IETF XML Registry

Этот документ регистрирует два URIs в реестре IETF XML Registry [RFC3688]. В соответствии с форматом [RFC3688] были выполнены приведенные ниже регистрации.

```
URI: urn:ietf:params:xml:ns:yang:ietf-datastores
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-origin
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.
```

## 8.2. Обновление реестра YANG Module Names Registry

Этот документ регистрирует два модуля YANG в реестре YANG Module Names [RFC6020]. В соответствии с форматом [RFC6020] были выполнены приведенные ниже регистрации.

```
name:         ietf-datastores
namespace:    urn:ietf:params:xml:ns:yang:ietf-datastores
prefix:       ds
reference:    RFC 8342

name:         ietf-origin
namespace:    urn:ietf:params:xml:ns:yang:ietf-origin
prefix:       or
reference:    RFC 8342
```

# 9. Вопросы безопасности

В этом документе обсуждается архитектурная модель для хранилищ данных сетевого управления, используемых протоколами NETCONF/RESTCONF и языком YANG. Это не оказывает влияния на безопасность Internet.

Хотя в этом документе заданы модули YANG, эти модули определяют лишь отождествления и аннотацию метаданных. По этой причине рекомендации по безопасности YANG [YANG-SEC] не используются.

Аннотация метаданных происхождения раскрывает источник значений в примененной конфигурации. Информация об источнике может подсказать, что в устройстве активны некоторые протоколы уровня управления. Поскольку информация об источнике связана с примененной конфигурацией, она доступна лишь для клиентов, имеющих полномочия считывать примененные конфигурационные параметры. Администраторам безопасности следует оценить конфиденциальность информации при определении правил контроля доступа.

# 10. Литература

## 10.1. Нормативные документы


[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126>.


## 10.2. Дополнительная литература


[NETMOD-Operational] Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", Work in Progress, draft-bjorklund-netmod-operational-00, October 2012.

[OpState-Enhance] Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", Work in Progress, draft-kwatsen-netmod-opstate-02, February 2016.

[OpState-Modeling] Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", Work in Progress, draft-openconfig-netmod-opstate-01, July 2015.

[OpState-Reqs] Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", Work in Progress, draft-ietf-netmod-opstate-reqs-04, January 2016.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <https://www.rfc-editor.org/info/rfc6244>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

[With-config-state] Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", Work in Progress, draft-wilton-netmod-opstate-yang-02, December 2015.

[YANG-SEC] IETF, "YANG Security Guidelines", <https://trac.ietf.org/trac/ops/wiki/yang-security-guidelines>.

# *Приложение A. Рекомендации по определению хранилищ*

Определение нового хранилища в рамках данной архитектуры следует приводить в специальном документе (например, RFC). Когда это имеет смысл, в одном документе может определяться множество хранилищ (например, когда эти хранилища логически связаны). В определении каждого хранилища должны включать описанные ниже аспекты.

## A.1. Определение модулей YANG, применимых для хранилища

Для хранилища данных могут быть применимы не все модули YANG. Некоторые хранилища могут вносить ограничения на использование моделей данных. Если для хранилища желательно целевое использование некоторого подмножества, документация должна указывать это.

## A.2. Определение применимого подмножества операторов YANG

По умолчанию данные в хранилище моделируются с использованием всех операторов YANG в доступных модулях YANG. Однако возможно задание критериев, которым должны удовлетворять операторы YANG для включения в хранилище. Например, могут разрешаться только узлы config true или узлы config false, имеющие конкретное расширение YANG.

## A.3. Определение способов актуализации данных

Для нового хранилища должны быть указаны способы его взаимодействия с существующими хранилищами данных.

Например, рисунок 2 показывает хранилища динамической конфигурации, подаваемые в хранилище <operational>. Способ такой передачи определяет конкретным хранилищем динамической конфигурации. В некоторых случаях это может происходить неявно просто при попадании данных в хранилище динамической конфигурации, а в других случаях может потребоваться явное действие (например, RPC).

## A.4. Определение применимых протоколов

По умолчанию предполагается что взаимодействие с хранилищами может выполняться с обоими протоколами NETCONF и RESTCONF. Однако можно задать использование лишь одного конкретного протокола (например, ForCES[1]), подмножества операций протокола или доступных возможностей (например, без блокировки или фильтрации по XPath).

## A.5. Определение отождествлений YANG для хранилища

Хранилище жлджно быть определено с использованием отождествления YANG, использующего ds:datastore или одно из производных от него отождествлений. Это отождествление требуется, чтобы по по нему можно было ссылаться на хранилище в операциях протокола (например, <get-data>).

Хранилище может быть также определено с использованием в качестве базы отождествления or:origin или производного от него отождествления. Такое отождествление требуется, если хранилище взаимодействует с <operational>, чтобы по нему можно указать хранилище в атрибуте метаданных origin, определенном в разделе 7.

Примеры использования этих рекомендаций приведены в Приложении B.

# *Приложение B. Пример эфемерного хранилища*

В этом приложении описаны примеры хранилища динамической конфигурации с использованием рекомендаций из Приложения A. Для краткости представлен сокращенный пример, предполагается что будет выпущен отдельный документ RFC с полным описанием.

Этот пример определяет хранилище динамической конфигурации с названием ephemeral, смоделированное на основе результатов рабочей группы I2RS.

[1]Forwarding and Control Element Separation - разделение элементов пересылки и управления.

| *Имя* | *Значение* |
|---|---|
| Name | ephemeral |
| YANG modules | all (default) |
| YANG nodes | all "config true" data nodes |
| How applied | changes automatically propagated to <operational> |
| Protocols | NETCONF/RESTCONF (default) |
| Defining YANG module | "example-ds-ephemeral" |

```
module example-ds-ephemeral {
  yang-version 1.1;
  namespace "urn:example:ds-ephemeral";
  prefix eph;

  import ietf-datastores {
    prefix ds;
  }
  import ietf-origin {
    prefix or;
  }

  // datastore identity
  identity ds-ephemeral {
    base ds:dynamic;
    description
      "The ephemeral dynamic configuration datastore.";
  }

  // origin identity
  identity or-ephemeral {
    base or:dynamic;
    description
      "Denotes data from the ephemeral dynamic configuration
       datastore.";
  }
}
```

# *Приложение C. Примеры*

Использование хранилищ является сложной задачей и многие тонкие эффекты проще показать на примерах. В этом приложении представлена серия примеров моделей данных с некоторым содержимым различных хранилищ.

Фрагменты XML [W3C.REC-xml-20081126] представлены лишь для иллюстрации.

## C.1. Пример хранилища System

Ниже показан используемый в примере функциональный модуль.

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

  import ietf-inet-types {
    prefix inet;
  }

  container system {
    leaf hostname {
      type string;
    }

    list interface {
      key name;

      leaf name {
        type string;
      }

      container auto-negotiation {
        leaf enabled {
          type boolean;
          default true;
        }
        leaf speed {
          type uint32;
```

```
            units mbps;
            description
              "The advertised speed, in Mbps.";
          }
        }

        leaf speed {
          type uint32;
          units mbps;
          config false;
          description
            "The speed of the interface, in Mbps.";
        }

        list address {
          key ip;

          leaf ip {
            type inet:ip-address;
          }
          leaf prefix-length {
            type uint8;
          }
        }
      }
    }
  }
```

Оператор настроил имя хоста и два интерфейса и хранилище <intended> имеет вид

```
  <system xmlns="urn:example:system">

    <hostname>foo.example.com</hostname>

    <interface>
      <name>eth0</name>
      <auto-negotiation>
        <speed>1000</speed>
      </auto-negotiation>
      <address>
        <ip>2001:db8::10</ip>
        <prefix-length>64</prefix-length>
      </address>
    </interface>

    <interface>
      <name>eth1</name>
      <address>
        <ip>2001:db8::20</ip>
        <prefix-length>64</prefix-length>
      </address>
    </interface>

  </system>
```

Система обнаружила, что аппаратная часть одного из настроенных интерфейсов (eth1) еще отсутствует, поэтому настройка данного интерфейса не была применена. После этого система получила имя хоста и дополнительный адрес IP для eth0 по протоколу DHCP. В дополнение к установке принятого по умолчанию значения для листа управления автоматическим согласованием (auto-negotiation) в системе также автоматически создан интерфейс loopback. Все упомянутое нашло отражение в хранилище <operational>. Отметим, что атрибут метаданных origin для некоторых узлов данных config true унаследован от их родителей.

```
  <system
      xmlns="urn:example:system"
      xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

    <hostname or:origin="or:learned">bar.example.com</hostname>

    <interface or:origin="or:intended">
      <name>eth0</name>
      <auto-negotiation>
        <enabled or:origin="or:default">true</enabled>
        <speed>1000</speed>
      </auto-negotiation>
      <speed>100</speed>
      <address>
        <ip>2001:db8::10</ip>
        <prefix-length>64</prefix-length>
      </address>
      <address or:origin="or:learned">
```

```
        <ip>2001:db8::1:100</ip>
        <prefix-length>64</prefix-length>
      </address>
    </interface>

    <interface or:origin="or:system">
      <name>lo0</name>
      <address>
        <ip>::1</ip>
        <prefix-length>128</prefix-length>
      </address>
    </interface>

  </system>
```

## C.2. Пример BGP

Рассмотрим фрагмент функционального модуля BGP

```
      container bgp {
        leaf local-as {
          type uint32;
        }
        leaf peer-as {
          type uint32;
        }
        list peer {
          key name;
          leaf name {
            type inet:ip-address;
          }
          leaf local-as {
            type uint32;
            description
              "... Defaults to ../local-as.";
          }
          leaf peer-as {
            type uint32;
            description
              "... Defaults to ../peer-as.";
          }
          leaf local-port {
            type inet:port;
          }
          leaf remote-port {
            type inet:port;
            default 179;
          }
          leaf state {
            config false;
            type enumeration {
              enum init;
              enum established;
              enum closing;
            }
          }
        }
      }
```

В этой модели оба узла bgp/peer/local-as и bgp/peer/peer-as имеют комплексные иерархические значения, позволяя пользователю задать используемые по умолчанию значения для всех партнеров в одном месте.

Модель также следует шаблону полного объединения узлов состояния (config false) и узлов конфигурации (config true). Здесь нет отдельной иерархии bgp-state и связанного с ней повтора узлов сдерживания ограничений и именования. Это делает модель более простой и удобочитаемой.

### *C.2.1. Хранилища данных*

Каждое хранилище дает разные представления этих узлов. Хранилище <running> будет содержать конфигурацию, заданную оператором (например, один партнер BGP). Хранилище <intended> концептуально будет содержать все проверенные на пригодность данные после исключения не предназначенных для такой проверки данных и выполнения всех локальных механизмов преобразования шаблонов. Хранилище <operational> будет показывать данные из <intended>, а также все узлы config false.

### *C.2.2. Добавление партнера*

Если уператор указал одного партнера BGP, этот партнер будет виден в обоих хранилищах <running> и <intended>. Он может также присутствовать в <candidate>, если сервер поддерживает хранилище для будущих конфигураций. Поиск партнера будет возвращать только заданные пользователем значения.

Между появлением партнера в хранилищах <running> и <intended> не должно быть задержки.

Добавим в хранилище <running> представленную ниже информацию.

```
<bgp>
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
  </peer>
</bgp>
```

#### C.2.2.1. Хранилище <operational>

Операционное хранилище будет содержать полностью раскрытые данные партнера, включая узлы config false. В нашем примере это означает появление узла state.

Кроме того, хранилище <operational> будет содержать реально используемые (currently in use) значения для всех узлов. Это значит, что local-as и peer-as будут запролнены даже в том случае, когда их значения не заданы в <intended>. При отсутствии bgp/peer/local-as будет использовано значение bgp/local-as, а при отсутствии bgp/peer/peer-as - bgp/peer-as. В операционном представлении это означает, что каждый партнер будет иметь значения для своих local-as и peer-as, даже если эти значения не будут заданы явно, но будут представлены в bgp/local-as и bgp/peer-as.

Каждый из партнеров BGP имеет связанное с ним соединение TCP, использующее значения local-port и remote-port из <intended>. Если эти значения не представлены, они будут выбраны системой. После организации соединения хранилище <operational> будет содержать текущие значения для local-port и remote-port, независимо от их происхождения. Если значения выбраны системой, атрибут origin будет указывать system. Перед организацией соединения один или оба узла могут отсутствовать, поскольку система может еще не иметь их значений.

```
<bgp xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
     or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>established</state>
  </peer>
</bgp>
```

### *C.2.3. Удаление партнера*

При изменении конфигурации может потребоваться время на прохождение этих изменений через вовлеченные в процесс программные компоненты. В этом интервале времени необходимо сохранять точное представление о работе устройства. Хранилище <operational> будет содержать узлы предыдущей и текущей конфигурации, насколько возможно точно отслеживая текущее операционное состояние устройства.

Рассмотрим сценарий с удалением партнера BGP. В этом случае операционное состояние будет по-прежнему отражать наличие этого партнера до тех пор, пока не будут освобождены ресурсы закрываемого партнерского соединения. В течение переходного периода текущие значения данных будут видны в хранилище <operational> с атрибутом origin, указывающим происхождение исходных данных.

```
<bgp xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
     or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>closing</state>
  </peer>
</bgp>
```

После освобождения ресурсов и закрытия соединения данные о партнере удаляются из хранилища <operational>.

## C.3. Пример интерфейса

В этом параграфе используется простая модель данных интерфейса.

```
container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
```

```
        }
      }
    }
```

### C.3.1. Заранее представленные интерфейсы

Одной из проблем сетевых устройств является поддержка сменных узлов (FRU[1]), которые могут добавляться и удаляться из устройства без его перезагрузки и нарушения нормальной работы. Эти FRU обычно являются интерфейсными модулями (платами) и устройства поддерживают упреждающее представление этих интерфейсов.

Если клиент создает интерфейс et-0/0/0, которого в этот момент нет физически, хранилище <intended> может содержать представленную ниже информацию.

```
<interfaces>
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
  </interface>
</interfaces>
```

Поскольку интерфейса нет, эти данные не будут присутствовать в хранилище <operational>.

При установке FRU с этим интерфейсом система обнаружит его и обработает соответствующую конфигурацию. Хранилище <operational> будет содержать данные из <intended>, а также узлы, добавленные системой типа текущего значения MTU для интерфейса.

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
    <mtu or:origin="or:system">1500</mtu>
  </interface>
</interfaces>
```

Если FRU удаляется, данные интерфейса исключаются из хранилища <operational>.

### C.3.2. Предоставляемые системой интерфейсы

Рассмотрим систему, предоставляющую петлевой интерфейс lo0 с принятым по умолчанию адресом IPv4 127.0.0.1 и IPv6 ::1. Система будет обеспечивать конфигурацию для этого интерфейса, если для него нет данных в <intended>.

При отсутствии конфигурации для lo0 в хранилище <intended>, <operational> будет показывать предоставляемые системой данные.

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface or:origin="or:system">
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Если конфигурация для lo0 имеется в <intended>, хранилище <operational> будет показывать эти данные с источником intended. Если значение ip-address не было задано, предоставленные системой значения будут иметь вид

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

## Благодарности

Этот документ является результатом многочисленных обсуждений, тянувшихся с 2010 года. Проблемы исходной модели хранилищ данных были рассмотрены в NETMOD-Operational] [With-config-state] [OpState-Reqs] [OpState-Enhance] [OpState-Modeling], а также [RFC6244]. Перечисленные ниже люди были авторами этих работ или внесли какой-либо иной вклад в создание этого документа.

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Andy Bierman, YumaWorks, <andy@yumaworks.com>
- Marcus Hines, Google, <hines@google.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Balazs Lengyel, Ericsson, <balazs.lengyel@ericsson.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Tom Petch, Engineering Networks Ltd, <ietfc@btconnect.com>
- Anees Shaikh, Google, <aashaikh@google.com>

[1]Field Replaceable Unit - заменяемый в процессе работы блок.

- Rob Shakir, Google, <robjs@google.com>
- Jason Sterne, Nokia, <jason.sterne@nokia.com>

Работа Juergen Schoenwaelder частично финансировалась в рамках Flamingo - проекта Network of Excellence (ICT-318488), поддерживаемого Европейской комиссией про программе Seventh Framework.

## *Адреса авторов*

**Martin Bjorklund**
Tail-f Systems
Email: mbj@tail-f.com

**Juergen Schoenwaelder**
Jacobs University
Email: j.schoenwaelder@jacobs-university.de

**Phil Shafer**
Juniper Networks
Email: phil@juniper.net

**Kent Watsen**
Juniper Networks
Email: kwatsen@juniper.net

**Robert Wilton**
Cisco Systems
Email: rwilton@cisco.com

Перевод на русский язык
Николай Малых
nmalykh@gmail.com