Internet Engineering Task Force (IETF) L. Lhotka
Request for Comments: 7951 CZ.NIC
Category: Standards Track August 2016
ISSN: 2070-1721

JSON Encoding of Data Modeled with YANG

Кодирование JSON для данных, моделируемых в YANG

Тезисы

Этот документ определяет правила кодирования для представления данных конфигурации и состояния, операций RPC[1] или действий, а также уведомлений, определенных с помощью языка YANG, в форме текста JSON[2].

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[3] и представляет согласованный взгляд сообщества IETF. Документ прошел 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/rfc7951.

Оглавление

1. Введение....2
2. Термины и обозначения....2
3. Свойства кодирования JSON....2
4. Имена и пространства имен....2
5. Кодирование экземпляров узлов данных YANG....3
   5.1. Узел leaf....4
   5.2. Узел container....4
   5.3. Узел leaf-list....4
   5.4. Узел list....4
   5.5. Узел anydata....5
   5.6. Узел anyxml....5
   5.7. Объекты метаданных....5
6. Представление типов YANG в значениях JSON....5
   6.1. Численные типы....5
   6.2. Тип string....5
   6.3. Тип boolean....5
   6.4. Тип enumeration....5
   6.5. Тип bits....6
   6.6. Тип binary....6
   6.7. Тип leafref....6
   6.8. Тип identityref....6
   6.9. Тип empty....6
   6.10. Тип union....6
   6.11. Тип instance-identifier....7
7. Соответствие I-JSON....7
8. Вопросы безопасности....7
9. Литература....7
   9.1. Нормативные документы....7
   9.2. Дополнительная литература....7
Приложение A. Полный пример....8
Благодарности....9
Адрес автора....9

[1]Remote Procedure Call - вызов удаленной процедуры.
[2]JavaScript Object Notation - обозначение объектов JavaScript.
[3]Internet Engineering Task Force.
[4]Internet Engineering Steering Group.

1. Введение

Протокол NETCONF[1] [RFC6241] использует XML [XML] для кодирования данных на своем уровне содержимого (Content Layer). Другие протоколы могут предпочесть другое кодирование, сохраняя преимущества YANG [RFC7950] для моделирования данных.

Например, протокол RESTCONF [RESTCONF] поддерживает кодирование XML (тип носителя application/yang.data+xml) и JSON (application/yang.data+json).

Спецификация языка моделирования YANG 1.1 [RFC7950] определяет только кодирование XML для деревьев данных, т. е. данных конфигурации и состояния, входных и выходных параметры операций RPC и действий, а также уведомлений. Цель этого документа заключается в определении правил кодирования тех же данных в форме текста JSON [RFC7159].

2. Термины и обозначения

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 [RFC2119].

Ниже приведены термины, определенные в [RFC7950].

- action - действие;
- anydata - любые данные;
- anyxml - любые данные XML;
- augment - дополнение;
- container - контейнер;
- data node - узел данных;
- data tree - дерево данных;
- identity - отождествление;
- instance identifier - идентификатор экземпляра;
- leaf - лист;
- leaf-list - лист-список;
- list - список;
- module - модуль;
- RPC operation - операция RPC;
- submodule - субмодуль.

Ниже приведены термины, определенные в [RFC6241].

- configuration data - данные конфигурации;
- notification - уведомление;
- state data - данные состояния.

3. Свойства кодирования JSON

Этот документ определяет кодирование JSON для деревьев YANG и их субдеревьев. Во всех случаях предполагается, что структура верхнего уровня с кодированием данных JSON является объектом.

Экземпляры улов данных YANG (leaf, container, leaf-list, list, anydata и anyxml) кодируются как элементы (member) объекта JSON, т. е. пары «имя - значение». В разделе 4 определено формирование имен, а в следующих разделах рассматриваются значения. Правила кодирования идентичны для всех типов деревьев данных, т. е. данных конфигурации и состояния, параметров операций RPC, действий и уведомлений.

За исключением кодирования anydata (параграф 5.5), правила этого документа применимы и в YANG 1.0 [RFC6020].

В отличие от содержимого элементов XML, значения JSON содержат лишь часть информации о типе (число, строка, логический). Кодирование JSON определено так, что эта информация никогда не противоречит типу данных соответствующего листа или листа списка YANG.

За исключением anyxml и узлов anydata без схемы, можно сопоставить дерево данных в кодировке JSON с кодировкой XML, определенной в [RFC7950], и наоборот. Однако для таких преобразования требуется модель данных YANG.

Для обеспечения максимальной совместимости реализаций, использующих разные анализаторы JSON, правила кодирования JSON, насколько это возможно, следуют ограничениям I-JSON (Internet JSON) профиля [RFC7493]. Более подробно соответствие I-JSON рассматривается в разделе 7.

4. Имена и пространства имен

Имя элемента объекта JSON **должно** иметь одну из приведенных ниже форм.

[1]Network Configuration Protocol - протокол настройки конфигурации сети.

- Простая (simple) форма совпадает с идентификатором соответствующего узла данных YANG.
- Форма с указанием пространства имен (namespace-qualified) использует перед идентификатором узла префикс в виде имени модуля, в котором был определен узел данных. Префикс отделяется от имени двоеточием (:).

Имя модуля определяет пространство имен для всех узлов данных, определенных в модуле. Если узел данных определен в субмодуле, в полном (namespace-qualified) имени элемента указывается имя основного модуля, к которому относится субмодуль.

Синтаксис ABNF [RFC5234] для имен элемента, показан на рисунке 1, а создание identifier определено в разделе 14 [RFC7950].

```
member-name = [identifier ":"] identifier
```

Рисунок 1. Создание ABNF для имени элемента JSON

Имя элемента с пространством имен **должно** применяться для всех элементов объекта JSON верхнего уровня, а также в тех случаях, когда пространства имен узла данных и его родителя отличаются. В остальных случаях **должна** применяться простая форма имен элементов.

Рассмотрим в качестве примера приведенный ниже модуль YANG.

```
module example-foomod {

  namespace "http://example.com/foomod";

  prefix "foomod";

  container top {
    leaf foo {
      type uint8;
    }
  }
}
```

Если модель данных включает лишь этот модуль, приведенное ниже JSON-кодирование данных конфигурации будет действительно.

```
{
  "example-foomod:top": {
    "foo": 54
  }
}
```

Отметим, что элемент объекта верхнего уровня использует полное имя, а для листа foo применяется простое, поскольку этот лист определен в одном модуле с родительским контейнером top.

Предположим, что контейнер top дополнен из другого модуля example-barmod, как показано ниже.

```
module example-barmod {

  namespace "http://example.com/barmod";

  prefix "barmod";

  import example-foomod {
    prefix "foomod";
  }

  augment "/foomod:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

Действительные конфигурационные данные в кодировке JSON, содержащие оба лисат, могут иметь вид

```
{
  "example-foomod:top": {
    "foo": 54,
    "example-barmod:bar": true
  }
}
```

Имя листа bar использует идентификатор пространства имен, поскольку его родитель определен в другом модуле.

Явные идентификаторы пространства имен иногда требуются при кодировании значений identityref и instance-identifier. В этом случае применяется определенная выше форма namespace-qualified. Более подробное описание дано в параграфах 6.8 и 6.11.

5. Кодирование экземпляров узлов данных YANG

Каждый экземпляр узла данных кодируется как пара «имя - значение», где имя формируется из идентификатора узла данных с использованием правил раздела 4. Значение зависит от категории узла данных, как описано ниже.

Для символов **должна** применяться кодировка UTF-8.

5.1. Узел leaf

Экземпляр листа (leaf) кодируется парой «имя - значение», где значением может быть строка, число, литерал true или false, а также специальный массив [null], в зависимости от типа листа (правила кодирования типов даны в разделе 6).

Ниже приведен пример для узла leaf.

```
leaf foo {
  type uint8;
}
```

Действительное кодирование JSON имеет вид

```
"foo": 123
```

5.2. Узел container

Экземпляр контейнера кодируется парой «имя - объект», а дочерние узлы контейнера - как элементы объекта.

Например, для определения контейнера

```
container bar {
  leaf foo {
    type uint8;
  }
}
```

Действительное кодирование JSON имеет вид

```
"bar": {
  "foo": 123
}
```

5.3. Узел leaf-list

Лист список кодируется парой «имя - массив» и элементами массива являются значения того или иного скалярного типа, которым может быть строка, число, литерал true или false, а также специальный массив [null], в зависимости от типа leaf-list (правила кодирования типов даны в разделе 6).

Порядок элементов массива следует тем же правилам, которые применяются для элементов XML, представляющих записи leaf-list при кодировании XML. В частности, **должны** соблюдаться свойства ordered-by (параграф 7.7.7 в [RFC7950]).

Например, для определения leaf-list

```
leaf-list foo {
  type uint8;
}
```

Действительное кодирование JSON имеет вид

```
"foo": [123, 0]
```

5.4. Узел list

Экземпляр списка (list) кодируется парой «имя - массив» и элементами массива являются объекты JSON.

Порядок элементов массива следует тем же правилам, которые применяются для элементов XML, представляющих записи списка при кодировании XML. В частности, **должны** соблюдаться свойства ordered-by (параграф 7.7.7 в [RFC7950]).

В отличии от кодирования XML, где ключи списка должны предшествовать всем «братским» (sibling) элементам в записи списка и сохранять порядок, заданный моделью данных, в записи списка с кодированием JSON может применяться произвольный порядок, поскольку объекты JSON являются неупорядоченным набором элементов.

Например, для определения списка

```
list bar {
  key foo;
  leaf foo {
    type uint8;
  }
  leaf baz {
    type string;
  }
}
```

Действительное кодирование JSON имеет вид

```
"bar": [
  {
    "foo": 123,
    "baz": "zig"
  },
  {
    "baz": "zag",
    "foo": 0
  }
]
```

5.5. Узел anydata

Узел anydata служит контейнером для произвольного набора узлов, которые иначе бы выглядели обычными данными модели YANG. Модель данных для содержимого anydata может (но не обязана) быть известной во время выполнения (runtime). При неизвестной модели преобразование экземпляров с кодированием JSON в кодирование XML, определенное в [RFC7950] может оказаться невозможным.

Экземпляр anydata кодируется как container, т. е. парой «имя - объект». Требование возможности моделирования содержимого anydata с помощью YANG предполагает перечисленные ниже правила для текста JSON внутри объекта.

- Это должен быть действительный текст I-JSON [RFC7493].
- Все имена элементов объекта должны соответствовать ABNF на рисунке 1.
- Любой массив JSON содержит лишь уникальные скалярные значения (как leaf-list, см. параграф 5.3) или только объекты (как list, см. параграф Section 5.4).
- Значение null разрешено лишь в одноэлементном массиве [null], соответствующем кодированию постого (empty) типа (см. параграф 6.9).

Например, для определения anydata

```
anydata data;
```

Действительное кодирование JSON имеет вид

```
"data": {
  "ietf-notification:notification": {
    "eventTime": "2014-07-29T13:43:01Z",
    "example-event:event": {
      "event-class": "fault",
      "reporting-entity": {
        "card": "Ethernet0"
      },
      "severity": "major"
    }
  }
}
```

5.6. Узел anyxml

Экземпляр anyxml кодируется в JSON парой «имя-значение» и значение **должно** соблюдать ограничения I-JSON.

Например, для определения anyxml

```
anyxml bar;
```

Действительное кодирование JSON имеет вид

```
"bar": [true, null, true]
```

5.7. Объекты метаданных

Помимо экземпляров узлов данных YANG документ JSON **может** включать специальные элементы, имена которых начинаются с символа @ (коммерческое AT). Такие элементы служат для специальных целей, таких как кодирование метаданных [RFC7952]. Точный синтаксис и семантика таких элементов выходят за рамки документа.

6. Представление типов YANG в значениях JSON

Тип значения JSON в экземпляре узла данных leaf или leaf-list зависит от типа этого узла данных, как описано в следующих параграфах.

6.1. Численные типы

Значения int8, int16, int32, uint8, uint16 и uint32 представляются числом JSON (number).

Значения int64, uint64 и decimal64 представляются строкой JSON, содержимое которой является лексическим представлением соответствующего типа YANG, как указано в параграфах 9.2.1 и 9.3.1 [RFC7950].

Например, если бы типом листа foo из параграфа 5.1 был uint64 вместо uint8, экземпляр бы имел вид

```
"foo": "123"
```

Специальная обработка 64-битовых чисел следует из рекомендаций I-JSON для кодирования чисел, выходящих за пределы диапазона двойной точности IEEE 754-2008 [IEEE754-2008], в форме строк (параграф 2.2 в [RFC7493]).

6.2. Тип string

Значение типа string представляется строкой JSON (string) в соответствии с правилами кодирования строк JSON.

6.3. Тип boolean

Логический тип (boolean) представляется в JSON литералом "true" или "false".

6.4. Тип enumeration

Перечисляемое значение (enumeration) представляются строкой JSON, которае являются одним из имен, назначенных оператором enum в YANG.

Это идентично лексическому представлению типа enumeration в XML (см. параграф 9.6 в [RFC7950]).

6.5. Тип bits

Значение bits представляется строкой JSON - последовательностью разделенных пробелами имен битов, которые нужно установить. Имена битов назначаются операторами bit в YANG.

Это идентично лексическому представлению типа bits, определенному в параграфе 9.7 [RFC7950].

6.6. Тип binary

Значение binary представляется строкой JSON - кодированием base64 для произвольных двоичных данных.

Это идентично лексическому представлению типа binary в XML (см. параграф 9.8 в [RFC7950]).

6.7. Тип leafref

Значение leafref представляется с использованием правил для типа leaf, на который указывает leafref.

6.8. Тип identityref

Значение identityref представляется строкой - именем отождествления. Если identity определено в разных модулях с злом leaf, содержащим значение identityref, **должна** применяться форма namespace-qualified (раздел 4). В остальных случаях разрешены обе формы.

Например, для представленного ниже семантического модуля

```
module example-mod {
  ...
  import ietf-interfaces {
    prefix if;
  }
  ...
  leaf type {
    type identityref {
      base "if:interface-type";
    }
  }
}
```

Действительный экземпляр листа type может иметь вид

```
"type": "iana-if-type:ethernetCsmacd"
```

Идентификатор пространства имен iana-if-type в этом случае требуется, поскольку отождествление ethernetCsmacd не определено в одном модуле с листом type.

6.9. Тип empty

Значение empty представляется как [null], т. е. массив с единственным литералом "null". В этом документе [null] считается неделимым (atomic) скалярным значением.

Это кодирование типа empty выбрано вместо использования просто "null" для упрощения работы с пустыми листьями в языках программирования общего назначения, где значение null для элемента считается отсутствием элемента.

Например, для определения листа

```
leaf foo {
  type empty;
}
```

Действительный экземпляр имеет вид

```
"foo": [null]
```

6.10. Тип union

Значение типа union кодируется как значение любого из входящих в объединение типов.

При проверке пригодности значения union информация о типе в кодировании JSON также **должна** учитываться. Синтаксис JSON обеспечивает способы преобразования типа члена объединения, которых нет в кодировании XML.

Например, рассмотрим приведенное ниже определение YANG.

```
leaf bar {
  type union {
    type uint16;
    type string;
  }
}
```

В RESTCONF [RESTCONF] можно установить значение bar с помощью типа носителя application/yang.data+xml

```
<bar>13.5</bar>
```

поскольку значение может интерпретироваться как строка, т. е. второй член объединения. Однако при использовании носителя application/yang.data+json будет ошибкой указать

```
"bar": 13.5
```

В этом случае кодирование JSON указывает, что значение должно быть строкой, а не числом и uint16 не пригодно.

И наоборот, значение

```
"bar": "1"
```

интерпретируется как строка.

6.11. Тип instance-identifier

Значение instance-identifier кодируется строкой аналогично лексическому представлению в XML (параграф 9.13.2 в [RFC7950]). Однако кодирование пространства имен instance-identifier следует правилам раздела 4.

- Имя самого левого (верхнего уровня) узла данных всегда имеет форму namespace-qualified.
- Имена других узлов данных имеют форму namespace-qualified, если узел и его родитель определены в разных моделях, иначе применяется простая форма. Правило относится и к именам узлов в предикатах.

Например,

```
/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip
```

указывает пригодное значение instance-identifier, поскольку узлы interfaces, interface и name определены в модуле ietf-interfaces, а ipv4 и ip - в модуле ietf-ip.

7. Соответствие I-JSON

I-JSON [RFC7493] задает профиль ограничений для JSON, гарантирующий максимальную совместимость протоколов, использующих JSON в своих сообщениях, независимо от применяемых в реализации кодеров и декодеров JSON. Определенное в этом документе кодирование по возможности следует требованиям и рекомендациям I-JSON .

В частности, гарантируются перечисленные ниже свойства.

- Кодировка символоа UTF-8.
- Уникальность имен элементов в одном объекте JSON.
- Независимость от порядка элементов объекта JSON.
- Надежное преобразование всех типов чисел, поддерживаемых YANG (см. параграф 6.1).

Определенное здесь кодирование JSON отличается от I-JSON только для представления типа binary. Для совместимости с кодированием XML применяется схема base64 (параграф 6.6), а I-JSON рекомендует base64url.

8. Вопросы безопасности

Этот документ определяет дополнительное кодирование для данных, моделируемых на языке YANG, и не добавляет проблем безопасности к рассмотренным в разделе 17 [RFC7950].

Документ не определяет механизмов подписывания или шифрования данных, моделируемых с помощью YANG. При обычных условиях безопасность и целостность данных обеспечивается используемыми протоколами управления, такими как NETCONF [RFC6241] и RESTCONF [RESTCONF]. Если этого не достаточно, следует рассмотреть другие механизмы, такие как криптография на основе открытых ключей PKCS[1] #7 [RFC2315] или JOSE[2] [RFC7515] [RFC7516].

Обработка JSON достаточно отличается от XML и анализаторы JSON могут быть подвержены отличающимся от анализаторов XML уязвимостям. Для снижения рисков безопасности программам на приемной стороне **следует** отвергать все сообщения, не соответствующие этому документу и передавать отправителям подходящие сообщения об ошибках.

9. Литература

9.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[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>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159[3], DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

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

9.2. Дополнительная литература

[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>.

[1]Public-Key Cryptography Standards - стандарты шифрования с открытыми ключами.
[2]JSON Object Signing and Encryption - подписывание и шифрование объектов JSON.
[3]Заменен RFC 8259. *Прим. перев.*

[RESTCONF] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", Work in Progress, draft-ietf-netconf-restconf-16[4], August 2016.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <http://www.rfc-editor.org/info/rfc2315>.

[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>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

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

[XML] 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, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Приложение A. Полный пример

Показанный ниже документ JSON представляет те же данные, которые приведены в качестве отклика на запрос NETCONF <get> в Приложении D к [RFC7223]. Модель данных объединяет два модуля YANG - ietf-interfaces и ex-vlan (пример модуля в Приложении C к [RFC7223]). Поддерживается свойство if-mib, определенное в модуле ietf-interfaces.

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ex-vlan:vlan-tagging": true
      },
      {
        "name": "eth1.10",
        "type": "iana-if-type:l2vlan",
        "enabled": true,
        "ex-vlan:base-interface": "eth1",
        "ex-vlan:vlan-id": 10
      },
      {
        "name": "lo1",
        "type": "iana-if-type:softwareLoopback",
        "enabled": true
      }
    ]
  },
  "ietf-interfaces:interfaces-state": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "admin-status": "down",
        "oper-status": "down",
        "if-index": 2,
        "phys-address": "00:01:02:03:04:05",
        "statistics": {
          "discontinuity-time": "2013-04-01T03:00:00+00:00"
        }
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "admin-status": "up",
        "oper-status": "up",
        "if-index": 7,
        "phys-address": "00:01:02:03:04:06",
```

[4]Документ опубликован в RFC 8040. *Прим. перев.*

```
          "higher-layer-if": [
            "eth1.10"
          ],
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "eth1.10",
          "type": "iana-if-type:l2vlan",
          "admin-status": "up",
          "oper-status": "up",
          "if-index": 9,
          "lower-layer-if": [
            "eth1"
          ],
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "eth2",
          "type": "iana-if-type:ethernetCsmacd",
          "admin-status": "down",
          "oper-status": "down",
          "if-index": 8,
          "phys-address": "00:01:02:03:04:07",
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        },
        {
          "name": "lo1",
          "type": "iana-if-type:softwareLoopback",
          "admin-status": "up",
          "oper-status": "up",
          "if-index": 1,
          "statistics": {
            "discontinuity-time": "2013-04-01T03:00:00+00:00"
          }
        }
      ]
    }
  }
```

Благодарности

Автор благодарен Andy Bierman, Martin Bjorklund, Dean Bogdanovic, Balazs Lengyel, Juergen Schoenwaelder и Phil Shafer за их полезные замечания и предложения.

Адрес автора

Ladislav Lhotka

CZ.NIC

Email: lhotka@nic.cz

Перевод на русский язык

Николай Малых

nmalykh@protocols.ru