

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 7, 2020

I. Petrov, Ed.
Acklio
November 04, 2019

YANG Object Universal Parsing Interface
draft-petrov-t2trg-youpi-01

Abstract

YANG Object Universal Parsing Interface (YOUPI) specification describes generic way to encode and decode binary data based on a YANG model for use of constrained devices. YOUPI is a generic mechanism designed for great flexibility, so that it can be adapted for any of the constainted devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Internet-Draft

YOUPI

November 2019

Table of Contents

1.	Introduction	2
1.1.	Terminology	3
2.	YOUPI	3
2.1.	YANG extentions	3
2.2.	Position	5
2.2.1.	Bit positions	5
2.2.2.	Cursor	5
2.2.3.	Absolute position	5
2.2.4.	Relative position	5
2.3.	FieldIndex	6
2.4.	Multiplier	6
2.5.	Offset	6
2.6.	Units-subject	6
2.7.	Data definitions	6
2.7.1.	Supported built-in type	6
2.7.2.	Leafs	7
2.7.3.	Type min/max values	7
2.7.4.	Type fraction digits	7
2.7.5.	Containers	8
2.7.6.	Condition	8
2.7.7.	Lists	9
2.7.8.	Enumerations as mappings	10
2.7.9.	Groupings	10
2.7.10.	Typedefs	10
3.	Security Considerations	10
4.	IANA Considerations	11
	Acknowledgements	11
	Contributors	11
7.	Normative References	11
Appendix A.	Complete examples	11
	Author's Address	11

[1.](#) Introduction

A huge number of very constraint IoT devices are expected to be coming to the market. They are very constraint in terms of the MTU (sometimes as small as 10b per message). As they are expected to be running for many years without the need for external energy, energy efficiency which is directly linked to the size of the payloads that need to be sent, is also very important. For those devices JSON and even CBOR formats might be too wasteful in terms of payload size.

The reality of the ecosystem is that currently a great number of applications use proprietary binary formats for exchanging information. A significant problem exists if those systems are to be interacting in a purely M2M fashion. While there are a number of possibilities to resolve those issues, due to the constraints it is

mandatory to have a way to extract and encode information from/to the binary payload and be able to annotate it with semantic metadata.

While binary formats can be rather complicated to parse and sometimes even context dependent (some entity needs to keep context in order to parse a message), for most cases a simple description format could be sufficient.

A good solution should not be bounded to the output format. It should be a data modeling language like YANG [[RFC7950](#)] that simply describes the structure of the obtained data and that allows different serialization formats afterwards.

[1.1](#). Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

[2](#). YOUPI

YOUPI provides a number of yang extentions as defined in [Section 2.1](#). Thanks to that additional information in the YANG definitions, it is possible to decode binary data and then transform it to a different easier to parse format like JSON, XML or CBOR. Additionally it defines extensions that allow meta information to be added so that JSON-LD is generated. This draft is not describing how the data is formatted as JSON or other format. For information how this could be done, please refer to RESTCONF, NETCONF or CORECONF.

The opposite process is also possible - generating binary packets from parsed data that comes from JSON or other format.

[2.1](#). YANG extentions

The definitions of the YANG extensions.

```
<CODE BEGINS> file "petrov-youpi-file@2019-07-22.yang"
module youpi {
    namespace "http://ackl.io/youpi";

    prefix "youpi";

    organization
        "Acklio";
```

Petrov

Expires May 7, 2020

[Page 3]

Internet-Draft

YOUPI

November 2019

```
contact
    "Ivaylo Petrov
    <mailto:ivaylo@ackl.io>";

description
    "This module defines the extentions used by youpi.";

revision 2019-07-22 {
    description "Initial revision.";
}

/**
 *
 * Extension for Binary data to CBOR mapping.
 *
 */
extension position {
    argument object;
}

extension fieldIndex {
    argument object;
}

extension condition {
    argument object;
}

extension multiplier {
```

```

        argument object;
    }

    extension offset {
        argument object;
    }

    extension units-subject {
        argument object;
    }

    extension js {
        argument object;
    }
}

```

[2.2.](#) Position

Information about which bits need to be used in order to find the value of a field.

[2.2.1.](#) Bit positions

If the position is not present or is empty, the value contains 0 bits and has a default value of 0 (or equivalent for the given type). Could be useful if a field needs to be the result of arithmetic operations from different fields.

It is possible to have a single bit read by giving only its value in the position extension.

If continuous bits need to be used to obtain the value of a given field, this can be achieved using the ".." syntax. For example "0..3" means bits 0, 1, 2 and 3.

If non-continuous bits need to be used, one can use the concatenation of bit ranges using the "|" operator. For example "0..1 | 3".

[2.2.2.](#) Cursor

Starts at 0 and changes with each read to the last bit index that was read. Used in [Section 2.2.4](#) to determine where the read will start from. [Section 2.2.3](#) is not affected by it, but changes its value.

[2.2.3.](#) Absolute position

The default one if no keyword is used. Alternatively "absolute" keyword can be provided to explicitly request such position.

Example:

```
leaf temp {
    type uint8;
    default -19;
    description "The temperature";
    youpi:position "0..6";
}
```

[2.2.4.](#) Relative position

Example:

```
leaf temp {
    type uint8;
    default -19;
    description "The temperature";
    youpi:position "relative 1..7";
}
```

This means that the value starts 1 bit after the current cursor and will read up to 7 bits after the current cursor position, including that 7th bit.

[2.3.](#) FieldIndex

Can be used to change the order in which fields are processed. By default the order in which fields appear in the document is the order

in which they are processed.

[2.4.](#) Multiplier

A value or another field by which a given field needs to be multiplied before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in [Section 2.5](#)).

[2.5.](#) Offset

A value or another field to which a given field needs to be added before the final value is obtained. The operations are executed in the order of appearance (this includes "offset" extension defined in [Section 2.5](#)).

[2.6.](#) Units-subject

Meta information used to compute JSON-LD.

[2.7.](#) Data definitions

[2.7.1.](#) Supported built-in type

- o binary
- o enumeration
- o int8
- o int16
- o int32

- o int64
- o string
- o uint8
- o uint16

- o uint32
- o uint64

[2.7.2.](#) Leafs

Simple fields like integers and strings are represented by leafs in YUUPI.

[2.7.3.](#) Type min/max values

"range" attribute can be used for giving a "min"/"max" acceptable value for a type. If the value is outside of the defined range, it is silently excluded from the final result.

Example:

```
typedef temp {  
    type int8 {  
        range "-20 .. 107";  
    }  
}
```

[2.7.4.](#) Type fraction digits

It is possible to specify how many fraction digits are expected for a value to have.

Example:

```
leaf temp {  
    type decimal64 {  
        fraction-digits 2;  
    }  
}
```

[2.7.5.](#) Containers

Complex fields like objects are represented by containers in YOUNPI.

[2.7.6.](#) Condition

[2.7.6.1.](#) Choice

Inside a choice statement, the condition extension gives information based on what value the choice will be decided.

For example considering that there is a value "mode" with the value of btn inside the model

```
leaf mode {  
    ...  
}  
choice data {  
    case _btn {  
        container button-data {  
            ...  
        }  
    }  
    case _temp {  
        container temperature-data {  
            ...  
        }  
    }  
    youpi:condition "../mode";  
}
```

Then the button-data container will be used to parse the data.

[2.7.6.2.](#) When

With when statement it is possible to link the presence of some piece of data to a value of another field. For example it is possible to have button-data or temperature-data depending of the value of the mode field.

```
container button-data {  
    when "../mode[.=1]"  
    ...  
}  
container temperature-data {  
    when "../mode[.=2]"  
    ...  
}
```

[2.7.7.](#) Lists

List statements are supported and they generate an array of a given composite type.

[2.7.7.1.](#) With explicit length

A list of minimum and maximum temperatures can be defined as:

```
leaf temperature-len {
    type int32;
}

list temperatures {
    youpi:length "../temperatures-len";
    leaf min-value {
        type int32;
    }
    leaf max-value {
        type int32;
    }
}
```

[2.7.7.2.](#) Until the end of input

The list as defined in [Section 2.7.7.1](#) can omit the length extension statement if all the remaining bytes in the payload are part of the list.

[2.7.7.3.](#) Until a specific value

The list as defined in [Section 2.7.7.1](#) can also omit the length if it has a defined key and if it only has one leaf or container in the list apart from the key and it is a subject to when statement that defines a stop value for the key.

```
list temperatures {
    key option-id;
    leaf option-id {
        type int32;
    }
    container value {
        when "../option-id[!=0xffffffff]";
        ...
    }
}
```

[2.7.8.](#) Enumerations as mappings

Enumerations can be used inside a typedef in order to restrict a field only to a set of acceptable values or in order to accomplish mapping between some values and other values (for example 0 stands for "temperature", 1 stands for "humidity", etc).

Example:

```
typedef mode-type {
    type enumeration {
        enum temp {
            value 0;
        }
        enum humidity {
            value 1;
        }
        enum light {
            value 2;
        }
        ...
    }
    ...
}
```

[2.7.9.](#) Groupings

Groupings can be used for better reuse of definitions. They don't affect the generated output.

[2.7.10.](#) Typedefs

Typedefs can be used to provide extra information about the type of a field, including semantic information about it.

[3.](#) Security Considerations

The YANG file should be valid.

Segmentation faults might result from invalid data being provided with a given YANG model.

Resource exhaustion can be looked for.

Petrov

Expires May 7, 2020

[Page 10]

Internet-Draft

YOUPI

November 2019

[4.](#) IANA Considerations

This document registers a YANG model.

Acknowledgements

Contributors

[7.](#) Normative References

- [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>>.
- [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>>.
- [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>>.

[Appendix A.](#) Complete examples

Author's Address

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

Petrov

Expires May 7, 2020

[Page 11]