IPFIX Working Group B. Trammell
Internet-Draft ETH Zurich
Intended status: Informational January 20, 2014
Expires: July 24, 2014

Textual Representation of IPFIX Abstract Data Types
draft-ietf-ipfix-text-adt-00.txt

Abstract

This document defines UTF-8 representations for IPFIX abstract data types, to support interoperable usage of the IPFIX Information Elements with protocols based on textual encodings.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The IPFIX Information Model, as defined by the IANA IPFIX Information Element Registry [iana-ipfix-assignments], provides a rich set of Information Elements for description of information about network entities and network traffic data, and abstract data types for these Information Elements. The IPFIX Protocol Specification [RFC7011], in turn, defines a big-endian binary encoding for these abstract data types suitable for use with the IPFIX Protocol.

However, present and future operations and management protocols and applications may use textual encodings, and generic framing and structure as in JSON or XML. A definition of canonical textual encodings for the IPFIX abstract data types would allow this set of Information Elements to be used for such applications, and for these applications to interoperate with IPFIX applications at the Information Element definition level.

Note that templating or other mechanisms for data description for such applications and protocols are application specific, and therefore out of scope for this document: only Information Element identification and data value representation are defined here.

2. Terminology

Capitalized terms defined in the IPFIX Protocol Specification [RFC7011] and the IPFIX Information Model [RFC7012] are used in this document as defined in those documents. In addition, this document defines the following terminology for its own use:

Enclosing Context
: Textual representation of IPFIX data values is applied to use the IPFIX Information Model within some existing textual format (e.g. XML, JSON). This outer format is referred to as the Enclosing Context within this document. Enclosing Contexts define escaping and quoting rules for represented data values.

3. Identifying Information Elements

The IPFIX Information Element Registry [iana-ipfix-assignments] defines a set of Information Elements and numbered by Information Element Identifiers, and named for human-readability. These Information Element Identifiers are meant for use with the IPFIX protocol, and have little meaning when applying the IPFIX Information Element Registry to textual representations.

Instead, applications using textual representations of Information Elements SHOULD use Information Element names to identify them; see Appendix A for examples illustrating this principle.

4. Data Type Encodings

Each subsection of this section defines a textual encoding for the abstract data types defined in [RFC7012]. This section uses ABNF [RFC5234], including the Core Rules in Appendix B, to describe the format of textual representations of IPFIX abstract data types.

4.1. octetArray

If the Enclosing Context defines a representation for binary objects, that representation SHOULD be used.

Otherwise, since the goal of textual representation of Information Elements is readability over compactness, the values of Information Elements of the octetArray data type are represented as a string of pairs of hexadecimal digits, one pair per byte, in the order the bytes would appear on the wire were the octetArray encoded directly in IPFIX per [RFC7011]. Whitespace may occur between any pair of digits to assist in human readability of the string, but is not necessary, and must be disregarded by any process reading the string. In ABNF:

```
hex-octet = 2HEXDIGIT

octetarray = 1* (hex-octet [WSP])
```

4.2. unsigned8, unsigned16, unsigned32, and unsigned64

If the Enclosing Context defines a representation for unsigned integers, that representation SHOULD be used.

In the special case that the unsigned Information Element has identifier semantics, and refers to a set of codepoints, either in an external registry, a sub-registry, or directly in the description of the Information Element, then the name or short description for that codepoint MAY be used to improve readability.

Otherwise, the values of Information Elements of an unsigned integer type may be represented either as unprefixed base-10 (decimal) strings, or as base-16 (hexadecimal) strings prefixed by '0x'; in ABNF:

```
unsigned = 1*DIGIT / '0x' 1*HEXDIG
```

Leading zeroes are allowed in either encoding, and do not signify base-8 (octal) encoding.

The encoded value must be in range for the corresponding abstract data type or Information Element. Out of range values should be interpreted as clipped to the implicit range for the Information Element as defined by the abstract data type, or to the explicit range of the Information Element if defined. Minimum and maximum values for abstract data types are shown in Table 1 below.

type	minimum	maximum
unsigned8	0	255
unsigned16	0	65536
unsigned32	0	4294967295
unsigned64	0	18446744073709551615

Table 1: Ranges for unsigned abstract data types

4.3. signed8, signed16, signed32, and signed64

If the Enclosing Context defines a representation for signed integers, that representation SHOULD be used.

Otherwise, the values of Information Elements of signed integer types should be represented as optionally-prefixed base-10 (decimal) strings. In ABNF:

```
sign = "+" / "-"

signed = [sign] 1*DIGIT
```

If the sign is omitted, it is assumed to be positive. Leading zeroes are allowed, and do not signify base-8 (octal) encoding.

The encoded value must be in range for the corresponding abstract data type or Information Element. Out of range values should be interpreted as clipped to the implicit range for the Information Element as defined by the abstract data type, or to the explicit range of the Information Element if defined. Minimum and maximum values for abstract data types are shown in Table 2 below.

type	minimum	maximum
signed8	-128	+127
signed16	-32768	+32767
signed32	-2147483648	+2147483647
signed64	-9223372036854775808	+9223372036854775807

Table 2: Ranges for signed abstract data types

4.4. float32 and float64

If the Enclosing Context defines a representation for floating point numbers, that representation SHOULD be used.

Otherwise, the values of Information Elements of float32 or float64 types are represented as an optionally sign-prefixed, optionally base-10 exponent-suffixed, floating point decimal number. In ABNF:

```
sign = "+" / "-"

exponent = 'e' 1*3DIGIT

right-decimal = '.' 0*DIGIT

mantissa = 1*DIGIT [right-decimal]

float = [sign] mantissa [exponent]
```

The expressed value is (mantissa * 10 ^ exponent). If the sign is omitted, it is assumed to be positive. If the exponent is omitted, it is assumed to be zero. Leading zeroes may appear in the mantissa and/or the exponent.

Minimum and maximum values for abstract data types are shown in Table 3below.

type	minimum abs(x)	maximum abs(x)
float32	5.877e-39	3.403e38
float64	1.1125e-308	+1.798e308

Table 3: Ranges for floating-point abstract data types

4.5. boolean

If the Enclosing Context defines a representation for boolean values, that representation SHOULD be used.

Otherwise, a true boolean value should be represented with the literal string 1, and a false boolean value with the literal string 0. In ABNF:

boolean-yes = "1"

boolean-no = "0"

boolean = boolean-yes / boolean-no

4.6. macAddress

MAC addresses are represented as IEEE 802 MAC-48 addresses, hexadecimal bytes, most significant byte first, separated by colons. In ABNF, using the hex-octet production from Section 4.1:

macaddress = hex-octet 5(":" hex-octet)

4.7. string

As Information Elements of the string type are simply UTF-8 encoded strings, they are represented directly, subject to the escaping and encoding rules of the Enclosing Context. If the Enclosing Context cannot natively represent UTF-8 characters, the escaping facility provided by the Enclosing Context must be used for non-representable characters. Additionally, strings containing characters reserved in

the Enclosing Context (e.g. markup characters, quotes) must be escaped or quoted according to the rules of the Enclosing Context.

4.8. dateTime*

Timestamp abstract data types are represented generally as in [RFC3339], with two important differences. First, all IPFIX timestamps are expressed in terms of UTC, so textual representations of these Information Elements are explictly in UTC as well. Time zone offsets are therefore not required or supported. Second, there are four timestamp abstract data types, separated by the precision which they can express. Fractional seconds must be omitted in dateTimeSeconds, expressed in milliseconds in dateTimeMilliseconds, and so on.

In ABNF, taken from [RFC3339] and modified:

```
date-fullyear   = 4DIGIT
date-month      = 2DIGIT  ; 01-12
date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31
time-hour       = 2DIGIT  ; 00-23
time-minute     = 2DIGIT  ; 00-59
time-second     = 2DIGIT  ; 00-58, 00-59, 00-60
time-msec       = "." 3*DIGIT
time-usec       = "." 6*DIGIT
time-nsec       = "." 9*DIGIT
partial-time    = time-hour ":" time-minute ":" time-second

datetimeseconds      = full-date "T" partial-time
datetimemilliseconds = full-date "T" partial-time "." time-msec
datetimemicroseconds = full-date "T" partial-time "." time-usec
datetimenanoseconds  = full-date "T" partial-time "." time-nsec
```

4.9. ipv4Address

IP version 4 addresses are represented in dotted-quad format, most-significant-byte first, as it would in a Uniform Resource Identifier [RFC3986]; the ABNF for an IPv4 address is taken from [RFC3986] and reproduced below:

```
dec-octet   = DIGIT                 ; 0-9
            / %x31-39 DIGIT         ; 10-99
            / "1" 2DIGIT            ; 100-199
            / "2" %x30-34 DIGIT     ; 200-249
            / "25" %x30-35          ; 250-255

ipv4address = dec-octet 3("." dec-octet)
```

4.10. ipv6Address

IP version 6 addresses are represented as in section 2.2 of [RFC4291], as updated by section 4 of [RFC5952]. The ABNF for an IPv6 address is taken from [RFC3986] and reproduced below:

```
ls32        = ( h16 ":" h16 ) / IPv4address
            ; least-significant 32 bits of address
h16         = 1*4HEXDIG
            ; 16 bits of address represented in hexadecimal
            ; zeroes to suppressed as in RFC 5952

ipv6address =                            6( h16 ":" ) ls32
            /                       "::" 5( h16 ":" ) ls32
            / [               h16 ] "::" 4( h16 ":" ) ls32
            / [ *1( h16 ":" ) h16 ] "::" 3( h16 ":" ) ls32
            / [ *2( h16 ":" ) h16 ] "::" 2( h16 ":" ) ls32
            / [ *3( h16 ":" ) h16 ] "::"    h16 ":"   ls32
            / [ *4( h16 ":" ) h16 ] "::"              ls32
            / [ *5( h16 ":" ) h16 ] "::"              h16
            / [ *6( h16 ":" ) h16 ] "::"
```

4.11. basicList, subTemplateList, and subTemplateMultiList

These abstract data types, defined for IPFIX Structured Data [RFC6313], do not represent actual data types; they are instead designed to provide a mechanism by which complex structure can be represented in IPFIX below the template level. It is assumed that protocols using textual Information Element representation will provide their own structure. Therefore, Information Elements of these Data Types MUST NOT be used in textual representations.

5. Security Considerations

This document does not present any additional security measures beyond those presented by [RFC7011].

6. IANA Considerations

This document has no considerations for IANA.

7. References

7.1. Normative References

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

[RFC7011] Claise, B., Trammell, B., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, September 2013.

[iana-ipfix-assignments] Internet Assigned Numbers Authority, , "IP Flow Information Export Information Elements (http://www.iana.org/assignments/ipfix/ipfix.xml)", November 2012.

7.2. Informative References

[RFC6313] Claise, B., Dhandapani, G., Aitken, P., and S. Yates, "Export of Structured Data in IP Flow Information Export (IPFIX)", RFC 6313, July 2011.

[RFC7012] Claise, B. and B. Trammell, "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, September 2013.

[RFC7013] Trammell, B. and B. Claise, "Guidelines for Authors and Reviewers of IP Flow Information Export (IPFIX) Information Elements", BCP 184, RFC 7013, September 2013.

Appendix A. Example

In this section, we examine an IPFIX Template and a Data Record defined by that Template, and show how that Data Record would be represented in JSON according to the specification in this document. Note that this is specifically NOT a recommendation for a particular representation, merely an illustration of the encodings in this document.

Figure 1 shows a Template in IESpec format as defined in section 10.1 of [RFC7013]. A Message containing this Template and a Data Record

is shown in Figure 2, and a corresponding JSON Object using the text format defined in this document is shown in Figure 3.

```
      flowStartMilliseconds(152)<dateTimeMilliseconds>[8]
      flowEndMilliseconds(153)<dateTimeMilliseconds>[8]
      octetDeltaCount(1)<unsigned64>[4]
      packetDeltaCount(2)<unsigned64>[4]
      sourceIPv6Address(27)<ipv4Address>[4]{key}
      destinationIPv6Address(28)<ipv4Address>[4]{key}
      sourceTransportPort(7)<unsigned16>[2]{key}
      destinationTransportPort(11)<unsigned16>[2]{key}
      protocolIdentifier(4)<unsigned8>[1]{key}
      tcpControlBits(6)<unsigned8>[1]
      flowEndReason(136)<unsigned8>[1]
```

Figure 1: Sample flow template (IPFIX)

```
             1         2         3         4         5         6
   0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0 2
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | 0x000a        | length 135    | export time 1352140263        | msg
  | sequence 0                    | domain 1                      | hdr
  | SetID 2       | length 52     | tid 256       | fields 11     | tmpl
  | IE 152        | length 8      | IE 153        | length 8      | set
  | IE 1          | length 4      | IE 2          | length 4      |
  | IE 27         | length 16     | IE 28         | length 16     |
  | IE 7          | length 2      | IE 11         | length 2      |
  | IE 4          | length 1      | IE 6          | length 1      |
  | IE 136        | length 1      | SetID 256     | length 83     | data
  | start time                                     1352140261135  | set
  | end time                                       1352140262880  |
  | octets                195383  | packets                   88  |
  | sip6                                                          |
  |                       2001:0db8:000c:1337:0000:0000:0000:0002 |
  | dip6                                                          |
  |                       2001:0db8:000c:1337:0000:0000:0000:0003 |
  | sp        80  | dp     32991  | prt 6 | tcp 19| fe 3  |
  +-------------------------------------------------------+
```

Figure 2: IPFIX message containing sample flow

```
{
    "flowStartMilliseconds": "2012-11-05T18:31:01.135",
    "flowEndMilliseconds": "2012-11-05T18:31:02.880",
    "octetDeltaCount": 195383,
    "packetDeltaCount": 88,
    "sourceIPv6Address": "2001:db8:c:1337::2",
    "destinationIPv6Address": "2001:db8:c:1337::3",
    "sourceTransportPort": 80,
    "destinationTransportPort": 32991,
    "protocolIdentifier": "tcp",
    "tcpControlBits": 19,
    "flowEndReason": 3
}
```


Figure 3: JSON object containing sample flow

Author's Address

Brian Trammell
Swiss Federal Institute of Technology Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Phone: +41 44 632 70 13
Email: trammell@tik.ee.ethz.ch