Internet Engineering Task Force (IETF)
Request for Comments: 7373
Category: Standards Track
ISSN: 2070-1721

B. Trammell
ETH Zurich

September 2014

Textual Representation of IP Flow Information Export (IPFIX) Abstract Data Types

Abstract

This document defines UTF-8 representations for IP Flow Information Export (IPFIX) abstract data types (ADTs) to support interoperable usage of the IPFIX Information Elements with protocols based on textual encodings.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7373.

Table of Contents

1. Introduction

The IP Flow Information Export (IPFIX) Information Model [RFC7012] provides a set of abstract data types (ADTs) for the IANA "IPFIX Information Elements" registry [IANA-IPFIX], which contains 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 [RFC7159] 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 used for data description for such applications and protocols are application specific and, therefore, out of scope for this document: only Information Element identification and value representation are defined here.

In most cases where a textual representation will be used, an explicit tradeoff is made for human readability or manipulability over compactness; this assumption is used in defining standard representations of IPFIX ADTs.

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. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In addition, this document defines the following terminology for its own use:

Enclosing Context
: A textual representation of Information Element values is applied to use the IPFIX Information Model within some existing textual format (e.g., XML [W3C-XML] and JSON [RFC7159]). This outer format is referred to as the Enclosing Context within this document. Enclosing Contexts define escaping and quoting rules for represented values.

3. Identifying Information Elements

The "IPFIX Information Elements" registry [IANA-IPFIX] defines a set of Information Elements 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 Elements" registry to textual representations.

Instead, applications using textual representations of Information Elements 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, including the Core Rules in Appendix B of [RFC5234], to describe the format of textual representations of IPFIX abstract data types.

If future documents update [RFC7012] to add new abstract data types to the IPFIX Information Model, and those abstract data types are generally useful, this document will also need to be updated in order to define textual encodings for those 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 human 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. In ABNF:

```
hex-octet = 2HEXDIG

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

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

Otherwise, the values of Information Elements of an unsigned integer type may be represented as either unprefixed base-10 (decimal) strings, base-16 (hexadecimal) strings prefixed by "0x", or base-2 (binary) strings prefixed by "0b". In ABNF:

```
unsigned = 1*DIGIT / "0x" 1*HEXDIG / "0b" 1*BIT
```

Leading zeroes are allowed in any representation and do not signify base-8 (octal) representation. Binary representation is intended for use with Information Elements with flag semantics, but it can be used in any case.

The encoded value MUST be in range for the corresponding abstract data type or Information Element. Values that are out of range are 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	65535
unsigned32	0	4294967295
unsigned64	0	18446744073709551615

Table 1: Ranges for Unsigned Abstract Data Types (in Decimal)

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 are 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 representation "-0" is explicitly allowed and is equal to zero.

The encoded value MUST be in range for the corresponding abstract data type or Information Element. Values that are out of range are to 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 (in Decimal)

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 optionally sign-prefixed, optionally base-10 exponent-suffixed, floating-point decimal numbers, as in [IEEE.754.2008]. The special strings "NaN", "+inf", and "-inf" represent "not a number", "positive infinity", and "negative infinity", respectively.

In ABNF:

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

exponent = "e" [sign] 1*3DIGIT

right-decimal = "." 1*DIGIT

mantissa = 1*DIGIT [right-decimal]

num = [sign] mantissa [exponent]

naninf = "NaN" / (sign "inf")

float = num / naninf
```

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. Values MUST be within range for single- or double-precision numbers as defined in [IEEE.754.2008]; finite values outside the appropriate range are to be interpreted as clamped to be within the range. Note that no more than three digits are required or allowed for exponents in this encoding due to these ranges.

Note that since this representation is meant for human readability, writers MAY sacrifice precision to use a more human-readable representation of a given value, at the expense of the ability to recover the exact bit pattern at the reader. Therefore, decoders MUST NOT assume that the represented values are exactly comparable for equality.

4.5. boolean

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

Otherwise, a true boolean value is represented by the literal string "true" and a false boolean value by the literal string "false". In ABNF:

```
boolean-true = "true"

boolean-false = "false"

boolean = boolean-true / boolean-false
```

4.6. macAddress

Media Access Control (MAC) addresses are represented as IEEE 802 MAC-48 addresses, hexadecimal bytes with the most significant byte first, separated by colons. In ABNF:

```
hex-octet = 2HEXDIG

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

4.7. string

As Information Elements of the string type are simply Unicode strings (encoded as UTF-8 when appearing in Data Sets in IPFIX Messages [RFC7011]), they are represented directly, using the Unicode encoding rules and quoting and escaping rules of the Enclosing Context.

If the Enclosing Context cannot natively represent Unicode characters, the escaping facility provided by the Enclosing Context MUST be used for nonrepresentable characters. Additionally, strings containing characters reserved in the Enclosing Context (e.g., control characters, markup characters, and quotes) MUST be escaped or quoted according to the rules of the Enclosing Context.

It is presumed that the Enclosing Context has sufficient restrictions on the use of Unicode to prevent the unsafe use of nonprinting and control characters. As there is no accepted solution for the processing and safe display of mixed-direction strings, mixed-direction strings should be avoided using this encoding. Note also that since this document presents no additional requirements for the normalization of Unicode strings, care must be taken when comparing strings using this encoding; direct byte-pattern comparisons are not sufficient for determining whether two strings are equivalent. See

[RFC6885] and [PRECIS] for more on possible unexpected results and related risks in comparing Unicode strings.

4.8. The dateTime ADTs

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 explicitly 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 that they can express. Fractional seconds are omitted in dateTimeSeconds, expressed in milliseconds in dateTimeMilliseconds, and so on.

In ABNF, taken from [RFC3339] and modified as follows:

```
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       = "." 3DIGIT
time-usec       = "." 6DIGIT
time-nsec       = "." 9DIGIT
full-date       = date-fullyear "-" date-month "-" date-mday
integer-time    = time-hour ":" time-minute ":" time-second

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

4.9. ipv4Address

IP version 4 addresses are represented in dotted-quad format, most significant byte first, as it would be 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, using the ipv4address production from the previous section:

```
ls32        = ( h16 ":" h16 ) / ipv4address
            ; least significant 32 bits of address
h16         = 1*4HEXDIG
            ; 16 bits of address represented in hexadecimal
            ; zeroes to be suppressed as in RFC 5952

ipv6address =                            6( h16 ":" ) ls32
            /                       "::" 5( h16 ":" ) ls32
            / [               h16 ] "::" 4( h16 ":" ) ls32
            / [     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

The security considerations for the IPFIX protocol [RFC7011] apply.

Implementations of decoders of Information Element values using these representations must take care to correctly handle invalid input, but the encodings presented here are not special in that respect.

The encoding specified in this document, and representations that may be built upon it, are specifically not intended for the storage of data. However, since storage of data in the format in which it is exchanged is a very common practice, and the ubiquity of tools for indexing and searching text significantly increases the ease of searching and the risk of privacy-sensitive data being accidentally indexed or searched, the privacy considerations in Section 11.8 of

[RFC7011] are especially important to observe when storing data using the encoding specified in this document that was derived from the measurement of network traffic.

When using representations based on this encoding to transmit or store network traffic data, consider omitting especially privacy-sensitive values by not representing the columns or keys containing those values, as in black-marker anonymization as discussed in Section 4 of [RFC6235]. Other anonymization techniques described in [RFC6235] may also be useful in these situations.

The encodings for all abstract data types other than 'string' are defined in such a way as to be representable in the US-ASCII character set and, therefore, should be unproblematic for all Enclosing Contexts. However, the 'string' abstract data type may be vulnerable to problems with ill-formed UTF-8 strings as discussed in Section 6.1.6 of [RFC7011]; see [UTF8-EXPLOIT] for background.

6. References

6.1. Normative References

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

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002, <http://www.rfc-editor.org/info/rfc3986>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[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, <http://www.rfc-editor.org/info/rfc7011>.

6.2. Informative References

[IANA-IPFIX]
IANA, "IPFIX Information Elements", <http://www.iana.org/assignments/ipfix/>.

[IEEE.754.2008]
Institute of Electrical and Electronics Engineers, "IEEE Standard for Floating-Point Arithmetic", IEEE Standard 754, August 2008.

[PRECIS] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation and Comparison of Internationalized Strings in Application Protocols", Work in Progress, draft-ietf-precis-framework-18, September 2014.

[RFC6235] Boschi, E. and B. Trammell, "IP Flow Anonymization Support", RFC 6235, May 2011, <http://www.rfc-editor.org/info/rfc6235>.

[RFC6313] Claise, B., Dhandapani, G., Aitken, P., and S. Yates, "Export of Structured Data in IP Flow Information Export (IPFIX)", RFC 6313, July 2011, <http://www.rfc-editor.org/info/rfc6313>.

[RFC6885] Blanchet, M. and A. Sullivan, "Stringprep Revision and Problem Statement for the Preparation and Comparison of Internationalized Strings (PRECIS)", RFC 6885, March 2013, <http://www.rfc-editor.org/info/rfc6885>.

[RFC7012] Claise, B. and B. Trammell, "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, September 2013, <http://www.rfc-editor.org/info/rfc7012>.

[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, <http://www.rfc-editor.org/info/rfc7013>.

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

[UTF8-EXPLOIT]
Davis, M. and M. Suignard, "Unicode Technical Report #36: Unicode Security Considerations", The Unicode Consortium, November 2012.

[W3C-XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation REC-xml, November 2008.

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 but merely an illustration of the encodings in this document; the quoting and formatting in the example are JSON specific.

Figure 1 shows a Template in Information Element Specifier (IESpec) format as defined in Section 10.1 of [RFC7013]; a corresponding JSON object representing a record defined by this template in the text format specified in this document is shown in Figure 2.

```
flowStartMilliseconds(152)<dateTimeMilliseconds>[8]
flowEndMilliseconds(153)<dateTimeMilliseconds>[8]
octetDeltaCount(1)<unsigned64>[4]
packetDeltaCount(2)<unsigned64>[4]
sourceIPv6Address(27)<ipv6Address>[16]{key}
destinationIPv6Address(28)<ipv6Address>[16]{key}
sourceTransportPort(7)<unsigned16>[2]{key}
destinationTransportPort(11)<unsigned16>[2]{key}
protocolIdentifier(4)<unsigned8>[1]{key}
tcpControlBits(6)<unsigned16>[2]
flowEndReason(136)<unsigned8>[1]
```

Figure 1: Sample Flow Template in IESpec Format

```
{
    "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 2: JSON Object Containing Sample Flow

Acknowledgments

Thanks to Paul Aitken, Benoit Claise, Andrew Feren, Juergen Quittek, David Black, and the IESG for their reviews and comments. Thanks to Dave Thaler and Stephan Neuhaus for discussions that improved the floating-point representation section. This work is materially supported by the European Union Seventh Framework Programme under grant agreement 318627 mPlane.

Author's Address

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

Phone: +41 44 632 70 13
EMail: ietf@trammell.ch