

Internet Engineering Task Force (IETF)
Request for Comments: 6313
Updates: [5102](#)
Category: Standards Track
ISSN: 2070-1721

B. Claise
G. Dhandapani
P. Aitken
S. Yates
Cisco Systems, Inc.
July 2011

Export of Structured Data in IP Flow Information Export (IPFIX)

Abstract

This document specifies an extension to the IP Flow Information Export (IPFIX) protocol specification in [RFC 5101](#) and the IPFIX information model specified in [RFC 5102](#) to support hierarchical structured data and lists (sequences) of Information Elements in data records. This extension allows definition of complex data structures such as variable-length lists and specification of hierarchical containment relationships between Templates. Finally, the semantics are provided in order to express the relationship among multiple list elements in a structured data record.

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/rfc6313>.

Copyright Notice

Copyright (c) 2011 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 (<http://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.

Table of Contents

1. Overview	5
1.1. IPFIX Documents Overview	5
1.2. Relationship between IPFIX and PSAMP	6
2. Introduction	6
2.1. The IPFIX Track	7
2.2. The IPFIX Limitations	8
2.3. Structured Data Use Cases	8
2.4. Specifications Summary	11
3. Terminology	11
3.1. New Terminology	12
3.2. Conventions Used in This Document	12
4. Linkage with the IPFIX Information Model	12
4.1. New Abstract Data Types	12
4.1.1. basicList	12
4.1.2. subTemplateList	12
4.1.3. subTemplateMultiList	12
4.2. New Data Type Semantic	13
4.2.1. List	13
4.3. New Information Elements	13
4.3.1. basicList	13
4.3.2. subTemplateList	13
4.3.3. subTemplateMultiList	13
4.4. New Structured Data Type Semantics	13
4.4.1. undefined	14
4.4.2. noneOf	14
4.4.3. exactlyOneOf	14
4.4.4. oneOrMoreOf	15
4.4.5. allof	16
4.4.6. ordered	16
4.5. Encoding of IPFIX Data Types	16
4.5.1. basicList	17
4.5.2. subTemplateList	19
4.5.3. subTemplateMultiList	21
5. Structured Data Format	25
5.1. Length Encoding Considerations	25
5.2. Recursive Structured Data	26
5.3. Structured Data Information Elements Applicability in Options Template Sets	26
5.4. Usage Guidelines for Equivalent Data Representations	27
5.5. Padding	29
5.6. Semantic	29
6. Template Management	33
7. The Collecting Process's Side	33

8. Defining New Information Elements Based on the New Abstract Data Types	34
9. Structured Data Encoding Examples	34
9.1. Encoding a Multicast Data Record with basicList	35
9.2. Encoding a Load-Balanced Data Record with a basicList	37
9.3. Encoding subTemplateList	38
9.4. Encoding subTemplateMultiList	41
9.5. Encoding an Options Template Set Using Structured Data	46
10. Relationship with the Other IPFIX Documents	51
10.1. Relationship with Reducing Redundancy	51
10.1.1. Encoding Structured Data Element Using Common Properties	51
10.1.2. Encoding Common Properties Elements with Structured Data Information Element	51
10.2. Relationship with Guidelines for IPFIX Testing	53
10.3. Relationship with IPFIX Mediation Function	54
11. IANA Considerations	54
11.1. New Abstract Data Types	54
11.1.1. basicList	54
11.1.2. subTemplateList	54
11.1.3. subTemplateMultiList	55
11.2. New Data Type Semantics	55
11.2.1. list	55
11.3. New Information Elements	55
11.3.1. basicList	55
11.3.2. subTemplateList	56
11.3.3. subTemplateMultiList	56
11.4. New Structured Data Semantics	56
11.4.1. undefined	56
11.4.2. noneOf	57
11.4.3. exactlyOneOf	57
11.4.4. oneOrMoreOf	57
11.4.5. allof	57
11.4.6. ordered	58
12. Security Considerations	58
13. References	58
13.1. Normative References	58
13.2. Informative References	58
14. Acknowledgements	59
Appendix A. Additions to XML Specification of IPFIX Information Elements and Abstract Data Types	60
Appendix B. Encoding IPS Alert Using Structured Data Information Elements	65

Table of Figures

Figure 1:	basicList Encoding	17
Figure 2:	basicList Encoding with Enterprise Number	18
Figure 3:	Variable-Length basicList Encoding (Length < 255 Octets)	18
Figure 4:	Variable-Length basicList Encoding (Length 0 to 65535 Octets)	19
Figure 5:	subTemplateList Encoding	19
Figure 6:	Variable-Length subTemplateList Encoding (Length < 255 Octets)	20
Figure 7:	Variable-Length subTemplateList Encoding (Length 0 to 65535 Octets)	21
Figure 8:	subTemplateMultiList Encoding	21
Figure 9:	Variable-Length subTemplateMultiList Encoding (Length < 255 Octets)	23
Figure 10:	Variable-Length subTemplateMultiList Encoding (Length 0 to 65535 Octets)	24
Figure 11:	Encoding basicList, Template Record	35
Figure 12:	Encoding basicList, Data Record, Semantic allOf	36
Figure 13:	Encoding basicList, Data Record with Variable-Length Elements, Semantic allOf	37
Figure 14:	Encoding basicList, Data Record, Semantic exactlyOneOf ..	38
Figure 15:	Encoding subTemplateList, Template for One-Way Delay Metrics	39
Figure 16:	Encoding subTemplateList, Template Record	40
Figure 17:	Encoding subTemplateList, Data Set	40
Figure 18:	Encoding subTemplateMultiList, Template for Filtering Attributes	44
Figure 19:	Encoding subTemplateMultiList, Template for Sampling Attributes	44
Figure 20:	Encoding subTemplateMultiList, Template for Flow Record ..	45
Figure 21:	Encoding subTemplateMultiList, Data Set	45
Figure 22:	PSAMP SSRI to Be encoded	48
Figure 23:	Options Template Record for PSAMP SSRI Using subTemplateMultiList	48
Figure 24:	PSAMP SSRI, Template Record for interface	49
Figure 25:	PSAMP SSRI, Template Record for linecard	49
Figure 26:	PSAMP SSRI, Template Record for linecard and interface ..	49
Figure 27:	Example of a PSAMP SSRI Data Record, Encoded Using a subTemplateMultiList	50
Figure 28:	Common and Specific Properties Exported Together [RFC5473]	51
Figure 29:	Common and Specific Properties Exported Separately According to [RFC5473]	52
Figure 30:	Common and Specific Properties Exported with Structured Data Information Element	52
Figure 31:	Encoding IPS Alert, Template for Target	67
Figure 32:	Encoding IPS Alert, Template for Attacker	68

Figure 33: Encoding IPS Alert, Template for Participant	68
Figure 34: Encoding IPS Alert, Template for IPS Alert	69
Figure 35: Encoding IPS Alert, Data Set	69

1. Overview

1.1. IPFIX Documents Overview

The IPFIX protocol [RFC5101] provides network administrators with access to IP Flow information.

The architecture for the export of measured IP Flow information out of an IPFIX Exporting Process to a Collecting Process is defined in the IPFIX architecture [RFC5470], per the requirements defined in RFC 3917 [RFC3917].

The IPFIX architecture [RFC5470] specifies how IPFIX Data Records and Templates are carried via a congestion-aware transport protocol from IPFIX Exporting Processes to IPFIX Collecting Processes.

IPFIX has a formal description of IPFIX Information Elements, their name, type, and additional semantic information, as specified in the IPFIX information model [RFC5102].

In order to gain a level of confidence in the IPFIX implementation, probe the conformity and robustness, and allow interoperability, the guidelines for IPFIX testing [RFC5471] present a list of tests for implementers of compliant Exporting Processes and Collecting Processes.

The Bidirectional Flow Export [RFC5103] specifies a method for exporting bidirectional flow (biflow) information using the IP Flow Information Export (IPFIX) protocol, representing each biflow using a single Flow Record.

"Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports" [RFC5473] specifies a bandwidth-saving method for exporting Flow or packet information, by separating information common to several Flow Records from information specific to an individual Flow Record: common Flow information is exported only once.

1.2. Relationship between IPFIX and PSAMP

The specification in this document applies to the IPFIX protocol specifications [RFC5101]. All specifications from [RFC5101] apply unless specified otherwise in this document.

The Packet Sampling (PSAMP) protocol [RFC5476] specifies the export of packet information from a PSAMP Exporting Process to a PSAMP Collecting Process. Like IPFIX, PSAMP has a formal description of its information elements, their name, type, and additional semantic information. The PSAMP information model is defined in [RFC5477].

As the PSAMP protocol specifications [RFC5476] are based on the IPFIX protocol specifications, the specifications in this document are also valid for the PSAMP protocol.

Indeed, the major difference between IPFIX and PSAMP is that the IPFIX protocol exports Flow Records while the PSAMP protocol exports Packet Reports. From a pure export point of view, IPFIX will not distinguish a Flow Record composed of several packets aggregated together from a Flow Record composed of a single packet. So the PSAMP export can be seen as a special IPFIX Flow Record containing information about a single packet.

2. Introduction

While collecting the interface counters every five minutes has proven to be useful in the past, more and more granular information is required from network elements for a series of applications: performance assurance, capacity planning, security, billing, or simply monitoring. However, the amount of information has become so large that, when dealing with highly granular information such as Flow information, a push mechanism (as opposed to a pull mechanism, such as Simple Network Management Protocol (SNMP)) is the only solution for routers whose primary function is to route packets. Indeed, polling short-lived Flows via SNMP is not an option: high-end routers can support hundreds of thousands of Flows simultaneously. Furthermore, in order to reduce the export bandwidth requirements, the network elements have to integrate mediation functions to aggregate the collected information, both in space (typically, from different linecards or different Exporters) and in time.

Typically, it would be beneficial if access routers could export Flow Records, composed of the counters before and after an optimization mechanism on the egress interface, instead of exporting two Flow Records with identical tuple information.

In terms of aggregation in time, let us imagine that, for performance assurance, the network management application must receive the performance metrics associated with a specific Flow, every millisecond. Since the performance metrics will be constantly changing, there is a new dimension to the Flow definition: we are not dealing anymore with a single Flow lasting a few seconds or a few minutes, but with a multitude of one millisecond sub-flows for which the performance metrics are reported.

Which current protocol is suitable for these requirements: push mechanism, highly granular information, and huge number of similar records? IPFIX, as specified in [RFC 5101](#) would give part of the solution.

2.1. The IPFIX Track

The IPFIX working group has specified a protocol to export Flow information [[RFC5101](#)]. This protocol is designed to export information about IP traffic Flows and related measurement data, where a Flow is defined by a set of key attributes (e.g., source and destination IP address, source and destination port).

The IPFIX protocol specification [[RFC5101](#)] specifies that traffic measurements for Flows are exported using a TLV (type, length, value) format. The information is exported using a Template Record that is sent once to export the {type, length} pairs that define the data format for the Information Elements in a Flow. The Data Records specify values for each Flow.

Based on the requirements for IP Flow Information Export (IPFIX) [[RFC3917](#)], the IPFIX protocol has been optimized to export Flow-related information. However, thanks to its Template mechanism, the IPFIX protocol can export any type of information, as long as the relevant Information Element is specified in the IPFIX information model [[RFC5102](#)], registered with IANA [[IANA-IPFIX](#)], or specified as an enterprise-specific Information Element. For each Information Element, the IPFIX information model [[RFC5102](#)] defines a numeric identifier, an abstract data type, an encoding mechanism for the data type, and any semantic constraints. Only basic, single-valued data types, e.g., numbers, strings, and network addresses, are currently supported.

2.2. The IPFIX Limitations

The IPFIX protocol specification [RFC5101] does not support the encoding of hierarchical structured data and arbitrary-length lists (sequences) of Information Elements as fields within a Template Record. As it is currently specified, a Data Record is a "flat" list of single-valued attributes. However, it is a common data modeling requirement to compose complex hierarchies of data types, with multiple occurrences, e.g., 0..* cardinality allowed for instances of each Information Element in the hierarchy.

A typical example is the MPLS label stack entries model. An early NetFlow implementation used two Information Elements to represent the MPLS label stack entry: a "label stack entry position" followed by a "label stack value". However, several drawbacks were discovered. Firstly, the Information Elements in the Template Record had to be imposed so that the position would always precede the value. However, some encoding optimizations are based on the permutation of Information Element order. Secondly, a new semantic intelligence, not described in the information model, had to be hard-coded in the Collecting Process: the label value at the position "X" in the stack is contained in the "label stack value" Information Element following by a "label stack entry position" Information Element containing the value "X". Therefore, this model was abandoned.

The selected solution in the IPFIX information model [RFC5102] is a long series of Information Elements: `mplsTopLabelStackSection`, `mplsLabelStackSection2`, `mplsLabelStackSection3`, `mplsLabelStackSection4`, `mplsLabelStackSection5`, `mplsLabelStackSection6`, `mplsLabelStackSection7`, `mplsLabelStackSection8`, `mplsLabelStackSection9`, `mplsLabelStackSection10`. While this model removes any ambiguity, it overloads the IPFIX information model with repetitive information. Furthermore, if `mplsLabelStackSection11` is required, IANA [IANA-IPFIX] will not be able to assign the new Information Element next to the other ones in the registry, which might cause some confusion.

2.3. Structured Data Use Cases

Clearly, the MPLS label stack entries issue can best be solved by using a real structured data type composed of ("label stack entry position", "label stack value") pairs, potentially repeated multiple times in Flow Records, since this would be the most efficient from an information model point of view.

Some more examples enter the same category: how to encode the list of output interfaces in a multicast Flow, how to encode the list of BGP Autonomous Systems (AS) in a BGP Flow, how to encode the BGP communities in a BGP Flow, etc.

The one-way delay passive measurement, which is described in the IPFIX applicability [RFC5472], is yet another example that would benefit from a structured data encoding. Assuming synchronized clocks, the Collector can deduce the one-way delay between two Observation Points from the following two Information Elements, collected from two different Observation Points:

- Packet arrival time: observationTimeMicroseconds [RFC5477]
- Packet ID: digestHashValue [RFC5477]

In practice, this implies that many pairs of (observationTimeMicroseconds, digestHashValue) must be exported for each Observation Point, even if Hash-Based Filtering [RFC5475] is used. On top of that information, if the requirement is to understand the one-way delay per application type, the 5-tuple (source IP address, destination IP address, protocol, source port, destination port) would need to be added to every Flow Record. Instead of exporting this repetitive 5-tuple, as part of every single Flow Record a Flow Record composed of a structured data type such as the following would save a lot of bandwidth:

5-tuple

```
{ observationTimeMicroseconds 1, digestHashValue 1 }  
{ observationTimeMicroseconds 2, digestHashValue 2 }  
{ observationTimeMicroseconds 3, digestHashValue 3 }  
{ ... , ... }
```

As a last example, here is a more complex case of hierarchical structured data encoding. Consider the example scenario of an IPS (Intrusion Prevention System) alert data structure containing multiple participants, where each participant contains multiple attackers and multiple targets, with each target potentially composed of multiple applications, as depicted below:

```
alert
  signatureId
  protocolIdentifier
  riskRating
  participant 1
    attacker 1
      sourceIPv4Address
      applicationId
    ...
    attacker N
      sourceIPv4Address
      applicationId
    target 1
      destinationIPv4Address
      applicationId 1
    ...
    applicationId n
  ...
  target N
    destinationIPv4Address
    applicationId 1
  ...
    applicationId n
  participant 2
  ...
```

To export this information in IPFIX, the data would need to be flattened (thus, losing the hierarchical relationships) and a new IPFIX Template created for each alert, according to the number of applicationId elements in each target, the number of targets and attackers in each participant, and the number of participants in each alert. Clearly, each Template will be unique to each alert, and a large amount of CPU, memory, and export bandwidth will be wasted creating, exporting, maintaining, and withdrawing the Templates. See [Appendix B](#) for a specific example related to this case study.

2.4. Specifications Summary

This document specifies an IPFIX extension to support hierarchical structured data and variable-length lists by defining three new Information Elements and three corresponding new abstract data types called `basicList`, `subTemplateList`, and `subTemplateMultiList`. These are defined in Sections 4.1 and 4.3.

The three Structured Data Information Elements carry some semantic information so that the Collecting Process can understand the relationship between the different list elements. The semantic in the Structured Data Information Elements is provided in order to express the relationship among the multiple top-level list elements. As an example, if a list is composed of the elements (A,B,C), the semantic expresses the relationship among A, B, and C, regardless of whether A, B, and C are individual elements or a list of elements.

It is important to note that whereas the Information Elements and abstract data types defined in the IPFIX information model [RFC5102] represent single values, these new abstract data types are structural in nature and primarily contain references to other Information Elements and to Templates. By referencing other Information Elements and Templates from an Information Element's data content, it is possible to define complex data structures such as variable-length lists and to specify hierarchical containment relationships between Templates. Therefore, this document prefers the more generic "Data Record" term to the "Flow Record" term.

This document specifies three new abstract data types, which are basic blocks to represent structured data. However, this document does not comment on all possible combinations of `basicList`, `subTemplateList`, and `subTemplateMultiList`. Neither does it limit the possible combinations.

3. Terminology

IPFIX-specific terminology used in this document is defined in Section 2 of the IPFIX protocol specification [RFC5101] and Section 3 of the PSAMP protocol specification [RFC5476]. As in [RFC5101], these IPFIX-specific terms have the first letter of a word capitalized when used in this document.

3.1. New Terminology

Structured Data Information Element

One of the Information Elements supporting structured data, i.e., the `basicList`, `subTemplateList`, or `subTemplateMultiList` Information Elements specified in [Section 4.3](#).

3.2. Conventions Used in This Document

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 [RFC 2119](#) [[RFC2119](#)].

4. Linkage with the IPFIX Information Model

As in the IPFIX protocol specification [[RFC5101](#)], the new Information Elements specified in [Section 4.3](#) MUST be sent in canonical format in network-byte order (also known as the big-endian byte ordering).

4.1. New Abstract Data Types

This document specifies three new abstract data types, as described below.

4.1.1. `basicList`

The type "`basicList`" represents a list of zero or more instances of any Information Element, primarily used for single-valued data types. Examples include a list of port numbers, a list of interface indexes, a list of AS in a BGP AS-PATH, etc.

4.1.2. `subTemplateList`

The type "`subTemplateList`" represents a list of zero or more instances of a structured data type, where the data type of each list element is the same and corresponds with a single Template Record. Examples include a structured data type composed of multiple pairs of ("MPLS label stack entry position", "MPLS label stack value"), a structured data type composed of performance metrics, and a structured data type composed of multiple pairs of IP address, etc.

4.1.3. `subTemplateMultiList`

The type "`subTemplateMultiList`" represents a list of zero or more instances of a structured data type, where the data type of each list element can be different and corresponds with different Template definitions. Examples include a structured data type composed of

multiple access-list entries, where entries can be composed of different criteria types.

4.2. New Data Type Semantic

This document specifies a new data type semantic, in addition to the ones specified in [Section 3.2](#) of the IPFIX information model [RFC5102], as described below.

4.2.1. List

A list represents an arbitrary-length sequence of zero or more structured data Information Elements, either composed of regular Information Elements or composed of data conforming to a Template Record.

4.3. New Information Elements

This document specifies three new Information Elements, as described below.

4.3.1. basicList

A basicList specifies a generic Information Element with a basicList abstract data type as defined in [Section 4.1.1](#) and list semantics as defined in [Section 4.2.1](#). Examples include a list of port numbers, a list of interface indexes, etc.

4.3.2. subTemplateList

A subTemplateList specifies a generic Information Element with a subTemplateList abstract data type as defined in [Section 4.1.2](#) and list semantics as defined in [Section 4.2.1](#).

4.3.3. subTemplateMultiList

A subTemplateMultiList specifies a generic Information Element with a subTemplateMultiList abstract data type as defined in [Section 4.1.3](#) and list semantics as defined in [Section 4.2.1](#).

4.4. New Structured Data Type Semantics

Structured data type semantics are provided in order to express the relationship among multiple list elements in a Structured Data Information Element. These structured data type semantics require a new IPFIX subregistry, as specified in the "IANA Considerations" section. The semantics are specified in the following subsections.

4.4.1. undefined

The "undefined" structured data type semantic specifies that the semantic of list elements is not specified and that, if a semantic exists, then it is up to the Collecting Process to draw its own conclusions. The "undefined" structured data type semantic, which is the default value, is used when no other structured data type semantic applies.

For example, a mediator that wants to translate IPFIX [RFC5101] into the export of structured data according to the specifications in this document doesn't know what the semantic is; it can only guess, as the IPFIX specifications [RFC5101] does not contain any semantic. Therefore, the mediator should use the "undefined" semantic.

4.4.2. noneOf

The "noneOf" structured data type semantic specifies that none of the elements are actual properties of the Data Record.

For example, a mediator might want to report to a Collector that a specific Flow is suspicious, but that it checked already that this Flow does not belong to the attack type 1, attack type 2, or attack type 3. So this Flow might need some further inspection. In such a case, the mediator would report the Flow Record with a basicList composed of (attack type 1, attack type 2, attack type 3) and the respective structured data type semantic of "noneOf".

Another example is a router that monitors some specific BGP AS-PATHs and reports if a Flow belongs to any of them. If the router wants to export that a Flow does not belong to any of the monitored BGP AS-PATHs, the router reports a Data Record with a basicList composed of (BGP AS-PATH 1, BGP AS-PATH 2, BGP AS-PATH 3) and the respective structured data type semantic of "noneOf".

4.4.3. exactlyOneOf

The "exactlyOneOf" structured data type semantic specifies that only a single element from the structured data is an actual property of the Data Record. This is equivalent to a logical XOR operation.

For example, if a Flow record contains a basicList of outgoing interfaces with the "exactlyOneOf" semantic, then it implies that the reported Flow only egressed from a single interface, although the Flow Record lists all of the possible outgoing interfaces. This is a typical example of a per destination load-balancing.

Another example is a mediator that must aggregate Data Records from different Observation Points and report an aggregated Observation Point. However, the different Observation Points can be specified by different Information Element types depending on the Exporter. For example:

Exporter1 Observation Point is characterized by the exporterIPv4Address, so a specific Exporter can be represented.

Exporter2 Observation Point is characterized by the exporterIPv4Address and a basicList of ingressInterface, so the Exporting Process can express that the observations were made on a series of input interfaces.

Exporter3 Observation Point is characterized by the exporterIPv4Address and a specific lineCardId, so the Exporting Process can express that the observation was made on a specific linecard.

If the mediator models the three different types of Observation Points with the three Template Records below:

Template Record 1: exporterIPv4Address
Template Record 2: exporterIPv4Address, basicList of
 ingressInterface
Template Record 3: exporterIPv4Address, lineCardId

then it can represent the aggregated Observation Point with a subTemplateMultiList and the semantic "exactlyOneOf". The aggregated Observation Point is modeled with the Data Records corresponding to either Template Record 1, Template Record 2, or Template Record 3 but not more than one of these. This implies that the Flow was observed at exactly one of the Observation Points reported.

4.4.4. oneOrMoreOf

The "oneOrMoreOf" structured data type semantic specifies that one or more elements from the list in the structured data are actual properties of the Data Record. This is equivalent to a logical OR operation.

Consider an example where a mediator must report an aggregated Flow (e.g., by aggregating IP addresses from IP prefixes), with an aggregated Observation Point. However, the different Observation Points can be specified by different Information Element types as described in [Section 4.4.2](#).

If the mediator models the three different types of Observation Points with the three Template Records below:

```
Template Record 1: exporterIPv4Address
Template Record 2: exporterIPv4Address, basicList of
                  ingressInterface
Template Record 3: exporterIPv4Address, lineCardId
```

then it can represent the aggregated Observation Point with a subTemplateMultiList and the semantic "oneOrMoreOf". The aggregated Observation Point is modeled with the Data Records corresponding to either Template Record 1, Template Record 2, or Template Record 3. This implies that the Flow was observed on at least one of the Observation Points reported, and potentially on multiple Observation Points.

4.4.5. allOf

The "allOf" structured data type semantic specifies that all of the list elements from the structured data are actual properties of the Data Record.

For example, if a Record contains a basicList of outgoing interfaces with the "allOf" semantic, then the observed Flow is typically a multicast Flow where each packet in the Flow has been replicated to each outgoing interface in the basicList.

4.4.6. ordered

The "ordered" structured data type semantic specifies that elements from the list in the structured data are ordered.

For example, an Exporter might want to export the AS10 AS20 AS30 AS40 BGP AS-PATH. In such a case, the Exporter would report a basicList composed of (AS10, AS20, AS30, AS40) and the respective structured data type semantic of "ordered".

4.5. Encoding of IPFIX Data Types

The following subsections define the encoding of the abstract data types defined in [Section 4.1](#). These data types may be encoded using either fixed- or variable-length Information Elements, as discussed in [Section 5.1](#). Like in the IPFIX specifications [[RFC5101](#)], all lengths are specified in octets.

4.5.1. basicList

The basicList Information Element defined in [Section 4.3.1](#) represents a list of zero or more instances of an Information Element and is encoded as follows:

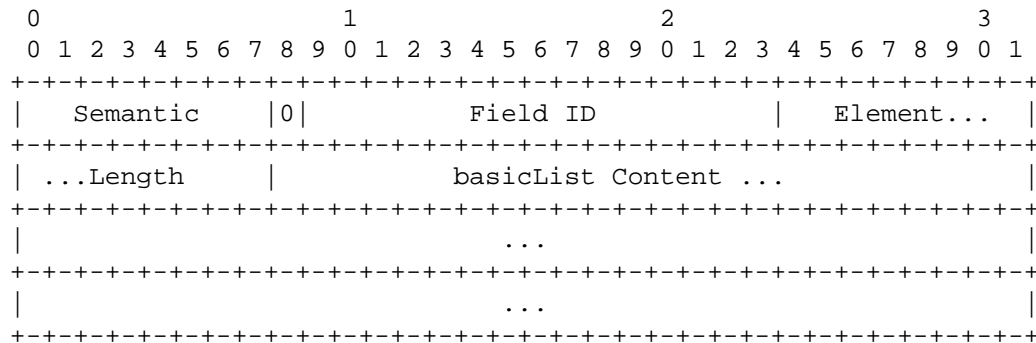


Figure 1: basicList Encoding

Semantic

The Semantic field indicates the relationship among the different Information Element values within this Structured Data Information Element. Refer to IANA's "IPFIX Structured Data Types Semantics" registry.

Field ID

Field ID is the Information Element identifier of the Information Element(s) contained in the list.

Element Length

Per [Section 7 of \[RFC5101\]](#), the Element Length field indicates the length, in octets, of each list element specified by Field ID, or contains the value 0xFFFF if the length is encoded as a variable-length Information Element at the start of the basicList Content.

Effectively, the Element Length field is part of the header, so even in the case of a zero-element list, it MUST NOT be omitted.

basicList Content

A Collecting Process decodes list elements from the basicList Content until no further data remains. A field count is not included but can be derived when the Information Element is decoded.

Note that in the diagram above, Field ID is shown with the Enterprise bit (most significant bit) set to 0. Instead, if the Enterprise bit is set to 1, a four-byte Enterprise Number MUST be encoded immediately after the Element Length as shown below. See the "Field Specifier Format" section in the IPFIX protocol [RFC5101] for additional information.

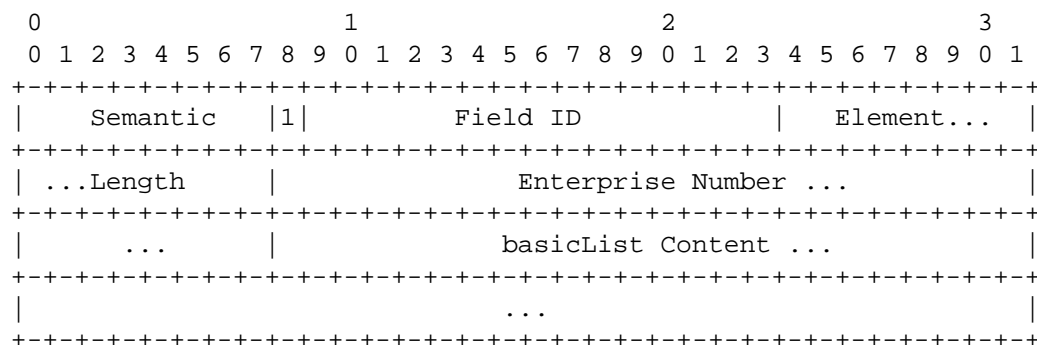


Figure 2: basicList Encoding with Enterprise Number

Also, note that if a basicList has zero elements, the encoded data contains the Semantic field, Field ID, the Element Length field, and the four-byte Enterprise Number (if present), while the basicList Content is empty.

If the basicList is encoded as a variable-length Information Element in less than 255 octets, it MAY be encoded with the Length field per Section 7 of [RFC5101] as shown in Figure 3. However, the three-byte length encoding, as shown in Figure 4, is RECOMMENDED (see Section 5.1).

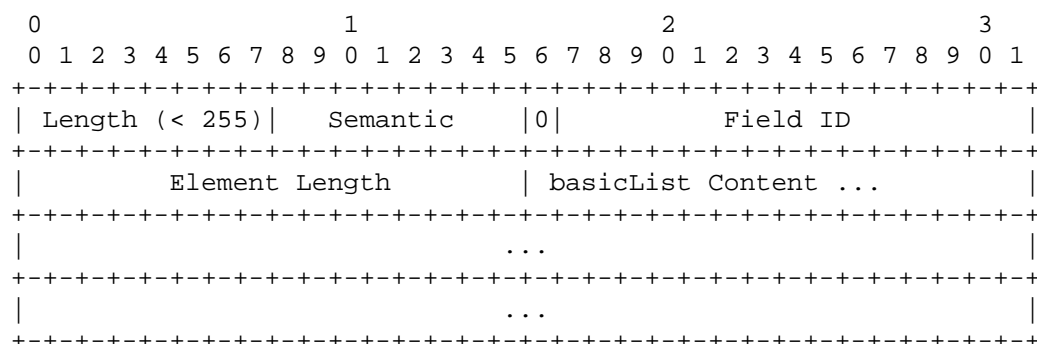


Figure 3: Variable-Length basicList Encoding
(Length < 255 Octets)

If the `basicList` is encoded as a variable-length Information Element in 255 or more octets, it MUST be encoded with the Length field per Section 7 of [RFC5101] as follows:

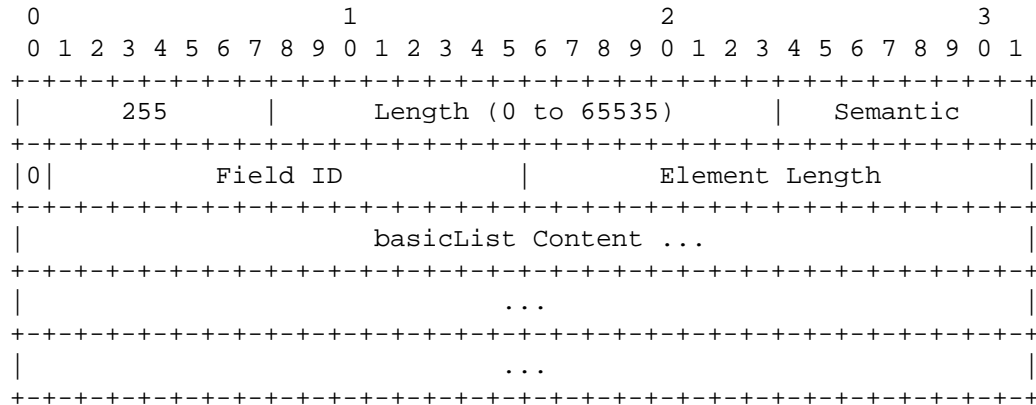


Figure 4: Variable-Length `basicList` Encoding
(Length 0 to 65535 Octets)

4.5.2. `subTemplateList`

The `subTemplateList` Information Element represents a list of zero or more Data Records corresponding to a specific Template. Because the Template Record referenced by a `subTemplateList` Information Element can itself contain other `subTemplateList` Information Elements, and because these Template Record references are part of the Information Elements content in the Data Record, it is possible to represent complex hierarchical data structures. The following diagram shows how a `subTemplateList` Information Element is encoded within a Data Record:

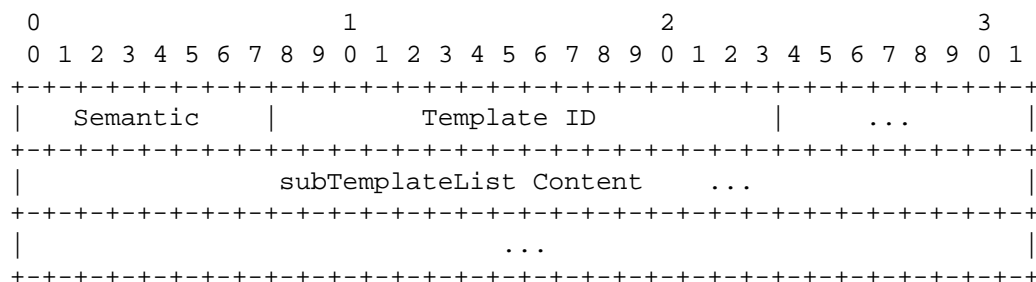


Figure 5: `subTemplateList` Encoding

Semantic

The Semantic field indicates the relationship among the different Data Records within this Structured Data Information Element.

Template ID

The Template ID field contains the ID of the Template used to encode and decode the subTemplateList Content.

subTemplateList Content

subTemplateList Content consists of zero or more instances of Data Records corresponding to the Template ID specified in the Template ID field. A Collecting Process decodes the subTemplateList Content until no further data remains. A record count is not included but can be derived when the subTemplateList is decoded. Encoding and decoding are performed recursively if the specified Template itself contains Structured Data Information Elements as described here.

Note that, if a subTemplateList has zero elements, the encoded data contains only the Semantic field and the Template ID field, while the subTemplateList Content is empty.

If the subTemplateList is encoded as a variable-length Information Element in less than 255 octets, it MAY be encoded with the Length field per [Section 7 of \[RFC5101\]](#) as shown in Figure 6. However, the three-byte length encoding, as shown in Figure 7, is RECOMMENDED (see [Section 5.1](#)).

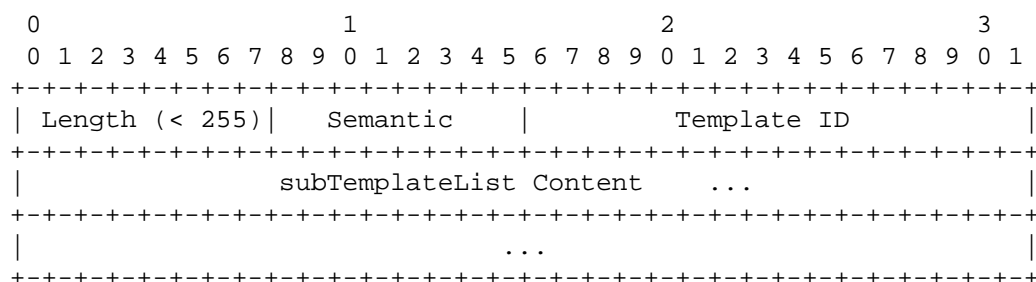


Figure 6: Variable-Length subTemplateList Encoding
(Length < 255 Octets)

If the subTemplateList is encoded as a variable-length Information Element in 255 or more octets, it MUST be encoded with the Length field per [Section 7 of \[RFC5101\]](#) as follows:

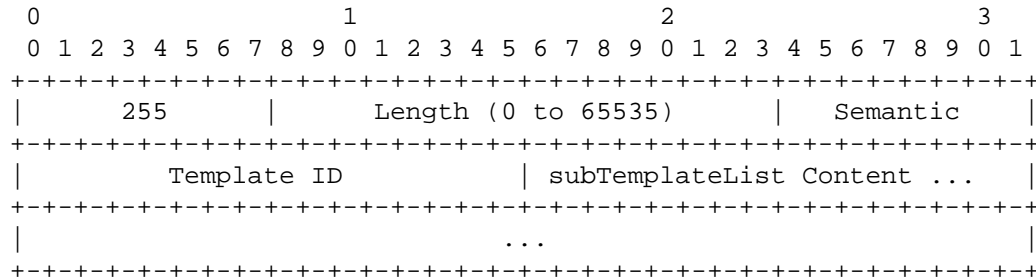
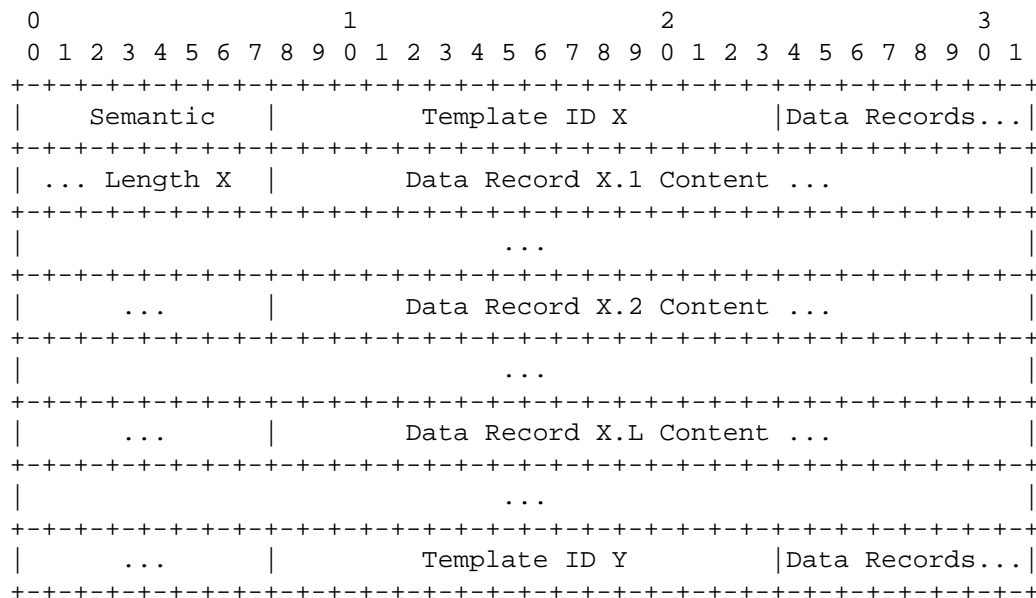


Figure 7: Variable-Length subTemplateList Encoding
(Length 0 to 65535 Octets)

4.5.3. subTemplateMultiList

Whereas each element in a subTemplateList Information Element corresponds to a single Template, it is sometimes useful for a list to contain elements corresponding to different Templates. To support this case, each top-level element in a subTemplateMultiList Information Element carries a Template ID, Length, and zero or more Data Records corresponding to the Template ID. The following diagram shows how a subTemplateMultiList Information Element is encoded within a Data Record. Note that the encoding following the Semantic field is consistent with the Set Header specified in [\[RFC5101\]](#).



```

| ... Length Y |          Data Record Y.1 Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |          Data Record Y.2 Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |          Data Record Y.M Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |          Template ID Z          |Data Records...|
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... Length Z |          Data Record Z.1 Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |          Data Record Z.2 Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |          Data Record Z.N Content ...          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     ...                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          ...          |
+-----+-----+-----+-----+-----+

```

Figure 8: subTemplateMultiList Encoding

Semantic

The Semantic field indicates the top-level relationship among the series of Data Records corresponding to the different Template Records within this Structured Data Information Element.

Template ID

Unlike the subTemplateList Information Element, each element of the subTemplateMultiList contains a Template ID that specifies the encoding of the following Data Records.

Data Records Length

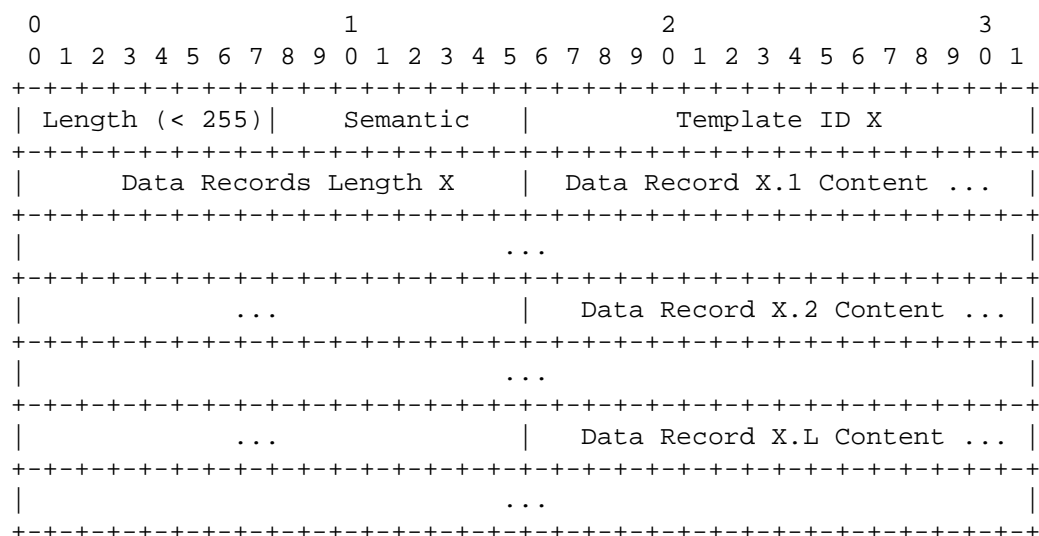
This is the total length of the Data Records encoding for the Template ID previously specified, including the two bytes for the Template ID and the two bytes for the Data Records Length field itself.

Data Record X.M

The Data Record X.M consists of the Mth Data Record of the Template Record X. A Collecting Process decodes the Data Records according to Template Record X until no further data remains, according to the Data Records Length X. Further Template IDs and Data Records may then be decoded according to the overall subTemplateMultiList length. A record count is not included but can be derived when the Element Content is decoded. Encoding and decoding are performed recursively if the specified Template itself contains Structured Data Information Elements as described here.

In the exceptional case of zero instances in the subTemplateMultiList, no data is encoded, only the Semantic field and Template ID field(s), and the Data Record Length field is set to zero.

If the subTemplateMultiList is encoded as a variable-length Information Element in less than 255 octets, it MAY be encoded with the Length field per [Section 7 of \[RFC5101\]](#) as shown in Figure 9. However, the three-byte length encoding, as shown in Figure 10, is RECOMMENDED (see [Section 5.1](#)).



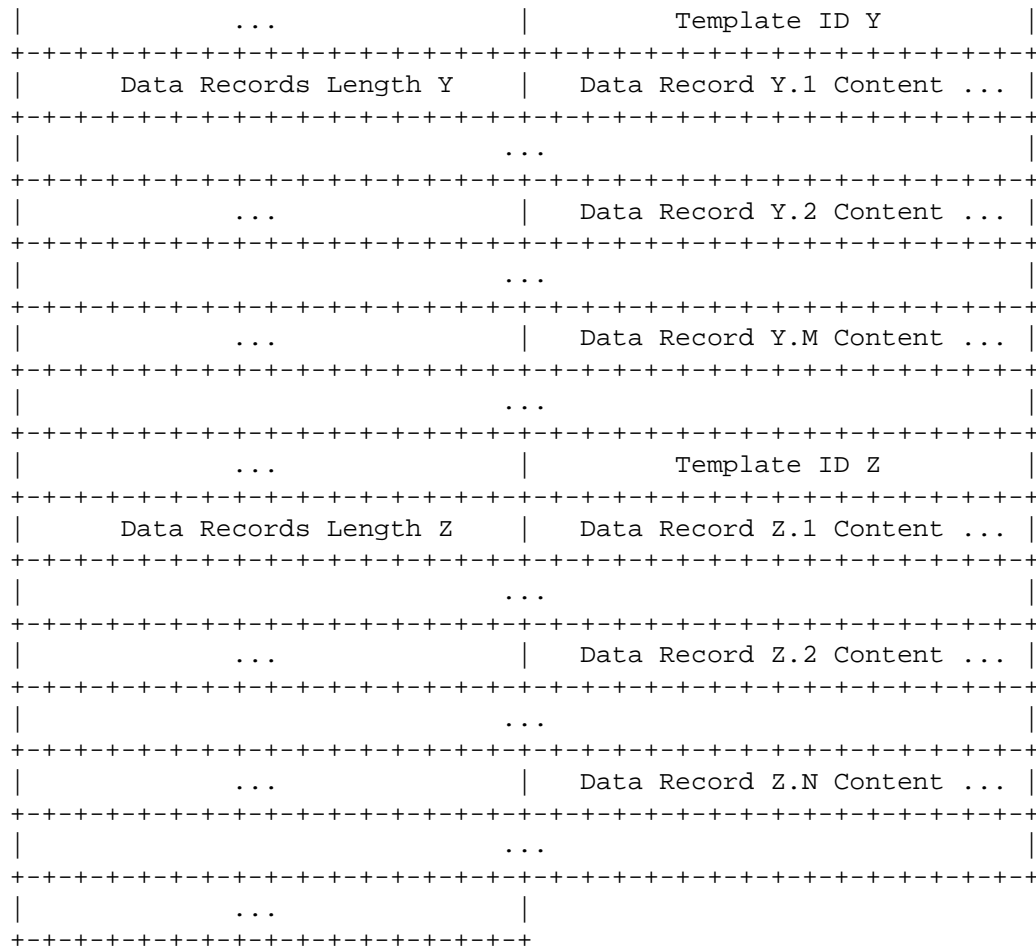
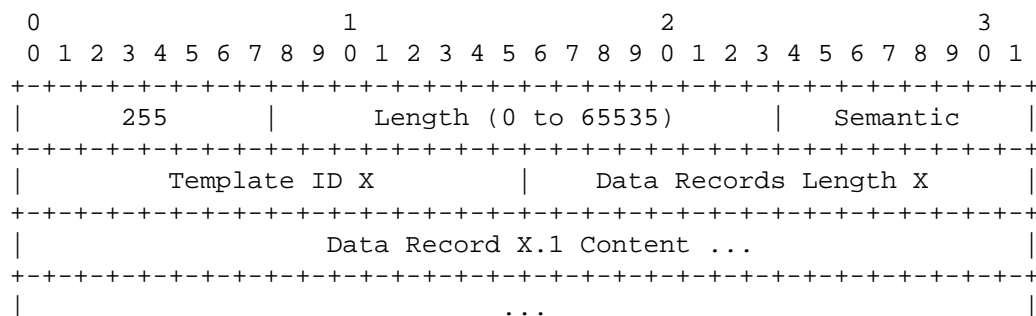


Figure 9: Variable-Length subTemplateMultiList Encoding
(Length < 255 Octets)

If the subTemplateMultiList is encoded as a variable-length Information Element in 255 or more octets, it MUST be encoded with the Length field per [Section 7 of \[RFC5101\]](#) as follows:




```

+-----+
|          Data Record X.2 Content ...          |
+-----+
|                      ...                      |
+-----+
|          Data Record X.L Content ...          |
+-----+
|                      ...                      |
+-----+
|      Template ID Y          |      Data Records Length Y      |
+-----+
|          Data Record Y.1 Content ...          |
+-----+
|                      ...                      |
+-----+
|          Data Record Y.2 Content ...          |
+-----+
|                      ...                      |
+-----+
|          Data Record Y.M Content ...          |
+-----+
|                      ...                      |
+-----+
|      Template ID Z          |      Data Records Length Z      |
+-----+
|          Data Record Z.1 Content ...          |
+-----+
|                      ...                      |
+-----+
|          Data Record Z.2 Content ...          |
+-----+
|                      ...                      |
+-----+
|          Data Record Z.N Content ...          |
+-----+
|                      ...                      |
+-----+

```

Figure 10: Variable-Length subTemplateMultiList Encoding
(Length 0 to 65535 Octets)

5. Structured Data Format

5.1. Length Encoding Considerations

The new Structured Data Information Elements represent a list that potentially carries complex hierarchical and repeated data.

When the encoding of a Structured Data Information Element has a fixed length (because, for example, it contains the same number of fixed-length elements, or if the permutations of elements in the list always produces the same total length), the element length can be encoded in the corresponding Template Record.

However, when representing variable-length data, hierarchical data, and repeated data with variable element counts, where the number and length of elements can vary from record to record, we RECOMMEND that the Information Elements are encoded using the variable-length encoding described in [Section 7 of \[RFC5101\]](#), with the length carried before the Structured Data Information Element encoding.

Because of the complex and repeated nature of the data, it is potentially difficult for the Exporting Process to efficiently know in advance the exact encoding size. In this case, the Exporting Process may encode the available data starting at a fixed offset and fill in the final length afterwards. Therefore, the three-byte length encoding is RECOMMENDED for variable-length Information Elements in all Template Records containing a Structured Data Information Element, even if the encoded length can be less than 255 bytes, because the starting offset of the data is known in advance.

When encoding such data, an Exporting Process MUST take care to not exceed the maximum allowed IPFIX message length of 65535 bytes as specified in [\[RFC5101\]](#).

5.2. Recursive Structured Data

It is possible to define recursive relationships between IPFIX structured data instances, for example, when representing a tree structure. The simplest case of this might be a `basicList`, where each element is itself a `basicList`, or a `subTemplateList` where one of the fields of the referenced Template is itself a `subTemplateList` referencing the same Template. Also, the Exporting Process MUST take care when encoding recursively-defined structured data not to exceed the maximum allowed length of an IPFIX Message (as noted in [Length Encoding Considerations](#)).

5.3. Structured Data Information Elements Applicability in Options Template Sets

Structured Data Information Elements MAY be used in Options Template Sets.

As an example, consider a mediation function that must aggregate Data Records from multiple Observation Point types:

Router 1, (interface 1)
Router 2, (linecard A)
Router 3, (linecard B)
Router 4, (linecard C, interface 2)

In order to encode the PSAMP Selection Sequence Report Interpretation [RFC5476], the mediation function must express this combination of Observation Points as a single new Observation Point. Recall from [RFC5476] that the PSAMP Selection Sequence Report Interpretation consists of the following fields:

Scope: selectionSequenceId
Non-Scope: one Information Element mapping the Observation Point
 selectorId (one or more)

Without structured data, there is clearly no way to express the complex aggregated Observation Point as "one Information Element mapping the Observation Point". However, the desired result may be easily achieved using the structured data types. Refer to [Section 9.5](#) for an encoding example related to this case study.

Regarding the scope in the Options Template Record, the IPFIX specification [RFC5101] mentions that "the IPFIX protocol doesn't prevent the use of any Information Elements for scope". Therefore, a Structured Data Information Element MAY be used as scope in an Options Template Set.

Extending the previous example, the mediation function could export a given name for this complex aggregated Observation Point:

Scope: Aggregated Observation Point (structured data)
Non-Scope: a new Information Element containing the name

5.4. Usage Guidelines for Equivalent Data Representations

Because `basicList`, `subTemplateList`, and `subTemplateMultiList` are all lists, in several cases, there is more than one way to represent what is effectively the same data structure. However, in some cases, one approach has an advantage over the other, e.g., more compact, uses fewer resources, and is therefore preferred over an alternate representation.

A `subTemplateList` can represent the same simple list of single-valued Information Elements as a `basicList`, if the Template referenced by the `subTemplateList` contains only one single-valued Information Element. Although the encoding is more compact than a `basicList` by two bytes, using a `subTemplateList`, in this case, requires a new

Template per Information Element. The basicList requires no additional Template and is therefore RECOMMENDED in this case.

Although a subTemplateMultiList with one Element can represent the contents of a subTemplateList, the subTemplateMultiList carries two additional bytes (Element Length). It is also potentially useful to a Collecting Process to know in advance that a subTemplateList directly indicates that list element types are consistent. The subTemplateList Information Element is therefore RECOMMENDED in this case.

The Semantic field in a subTemplateMultiList indicates the top-level relationship among the series of Data Records corresponding to the different Template Records, within this Structured Data Information Element. If a semantic is required to describe the relationship among the different Data Records corresponding to a single Template ID within the subTemplateMultiList, then an encoding based on a basicList of subTemplateLists should be used; refer to [Section 5.6](#) for more information. Alternatively, if a semantic is required to describe the relationship among all Data Records within a subTemplateMultiList (regardless of the Template Record), an encoding based on a subTemplateMultiList with one Data Record corresponding to a single Template ID can be used.

Note that the referenced Information Element(s) in the Structured Data Information Elements can be taken from the IPFIX information model [[RFC5102](#)], the PSAMP information model [[RFC5477](#)], any of the Information Elements defined in the IANA IPFIX registry [[IANA-IPFIX](#)], or enterprise-specific Information Elements.

If a Template Record contains a subTemplateList as the only field, a Set encoding as specified in the IPFIX protocol specifications [[RFC5101](#)] should be considered, unless:

- A relationship among multiple list elements must be exported, in which case, the semantic from the IPFIX Structured Data Information Element can convey this relationship.
- The Exporting Process wants to convey the number of elements in the list, even in the special cases of zero or one element in the list. Indeed, the case of an empty list cannot be represented with the IPFIX protocol specifications [[RFC5101](#)]. In the case of a single element list, the Template Record specified in the IPFIX protocol specification [[RFC5101](#)] could be used. However, on the top of the Template Record with the subTemplateList to export multiple list elements, this supplementary Template would impose some extra

management, both on the Exporting Process and on the Collecting Process, which might have to correlate the information from two Template Records.

Similarly, if a Template Record contains a subTemplateMultiList as the only field, an IPFIX Message as described in the IPFIX protocol specification [RFC5101] should be considered, unless:

- A relationship among top-level list elements must be exported, in which case, the semantic from the IPFIX Structured Data Information Element can convey this relationship.
- The Exporting Process wants to convey the number of Data Records corresponding to every Template in the subTemplateMultiList.

5.5. Padding

The Exporting Process MAY insert some padding octets in structured data field values in a Data Record by including the 'paddingOctets' Information Element as described in [RFC5101], Section 3.3.1. The paddingOctets Information Element can be included in a Template Record referenced by a structured data Information Element for this purpose.

5.6. Semantic

Semantic interpretations of received Data Records at or beyond the Collecting Process remain explicitly undefined, unless that data is transmitted using this extension with explicit structured data type semantic information.

It is not the Exporter's role to check the validity of the semantic representation of Data Records.

More complex semantics can be expressed as a combination of the Semantic Data Information Elements specified in this document.

For example, the export of the AS10 AS20 AS30 AS40 {AS50,AS60} BGP AS-PATH would be reported as a basicList of two elements, each element being a basicList of BGP AS, with the top-level structured data type semantic of "ordered". The first element would contain a basicList composed of (AS10,AS20,AS30,AS40) and the respective structured data type semantic of "ordered", while the second element would contain a basicList composed of (AS50, AS60) and the respective structured data type semantic of "exactlyOneOf". A high-level Data Record diagram would be represented as:

```
BGP AS-PATH = (basicList, ordered,  
               (basicList, ordered, AS10,AS20,AS30,AS40),  
               (basicList, exactlyOneOf, AS50, AS60)  
            )
```

If a semantic is required to describe the relationship among the different Data Records corresponding to a single Template ID within the subTemplateMultiList, then an encoding based on a basicList of subTemplateLists should be used, as shown in the next case study.

Case study 1:

In this example, an Exporter monitoring security attacks must export a list of security events consisting of attackers and targets. For the sake of the example, assume that the Collector can differentiate the attacker (which is expressed using source fields) from the target (which is expressed using destination fields). Imagine that attackers A1 or A2 may attack targets T1 and T2.

The first case uses a subTemplateMultiList composed of two Template Records, one representing the attacker and one representing the target, each of them containing an IP address and a port.

```
Attacker Template Record = (src IP address, src port)
```

```
Target Template Record = (dst IP address, dst port)
```

A high-level Data Record diagram would be represented as:

```
Alert = (subTemplateMultiList, allof,  
        (Attacker Template Record, A1, A2),  
        (Target Template Record, T1, T2)  
      )
```

The Collecting Process can only conclude that the list of attackers (A1, A2) and the list of targets (T1, T2) are present, without knowing the relationship amongst attackers and targets. The Exporting Process would have to explicitly call out the relationship amongst attackers and targets as the top-level semantic offered by the subTemplateMultiList isn't sufficient.

The only proper encoding for the previous semantic (i.e., attacker A1 or A2 may attack target T1 and T2) uses a basicList of subTemplateLists and is represented as follows:

```
Attacker Template Record = (src IP address, src port)

Target Template Record = (dst IP address, dst port)

Alert = (basicList, allof,

        (subTemplateList, exactlyOneOf, attacker A1, A2)

        (subTemplateList, allof, target T1, T2)

)
```

Case study 2:

In this example, an Exporter monitoring security attacks must export a list of attackers and targets. For the sake of the example, assume that the Collector can differentiate the attacker (which is expressed using source fields) from the target (which is expressed using destination fields). Imagine that attacker A1 or A2 is attacking target T1, while attacker A3 is attacking targets T2 and T3. The first case uses a subTemplateMultiList that contains Data Records corresponding to two Template Records, one representing the attacker and one representing the target, each of them containing an IP address and a port.

```
Attacker Template Record = (src IP address, src port)
Target Template Record = (dst IP address, dst port)
```

A high-level Data Record diagram would be represented as:

```
Alert = (subTemplateMultiList, allof,

        (Attacker Template Record, A1, A2, A3),

        (Target Template Record, T1, T2, T3)

)
```

The Collecting Process can only conclude that the list of attackers (A1, A2, A3), and the list of targets (T1, T2, T3) are present, without knowing the relationship amongst attackers and targets.

The second case could use a Data Record definition composed of the following:

```
Alert = (subTemplateMultiList, allof,  
        (Attacker Template Record, A1, A2),  
        (Target Template Record, T1),  
        (Attacker Template Record, A3),  
        (Target Template Record, T2, T3)  
    )
```

With the above representation, the Collecting Process can infer that the alert consists of the list of attackers (A1, A2), target (T1), attacker (A3), and list of targets (T2, T3). From the sequence in which attackers and targets are encoded, the Collector can possibly deduce that some relationship exists among (A1, A2, T1) and (A2, T1, T2) but cannot understand what it is exactly. So, there is a need for the Exporting Process to explicitly define the relationship between the attackers, and targets and the top-level semantic of the subTemplateMultiList is not sufficient.

The only proper encoding for the previous semantic (i.e., attacker A1 or A2 attacks target T1, attacker A3 attacks targets T2 and T3) uses a basicList of subTemplateLists and is represented as follows:

```
Participant P1 =  
  
    (basicList, allof,  
        (subTemplateList, exactlyOneOf, attacker A1, A2)  
        (subTemplateList, undefined, target T1)  
    )  
  
Participant P2 =  
  
    (basicList, allof,  
        (subTemplateList, undefined, attacker A3,  
        (subTemplateList, allof, targets T2, T3)  
    )
```


The security alert is represented as a subTemplateList of participants.

Alert =

(subTemplateList, allOf, Participant P1, Participant P2)

Note that, in the particular case of a single element in a Structured Data Information Element, the Semantic field is actually not very useful since it specifies the relationship among multiple elements. Any choice of allOf, exactlyOneOf, or OneOrMoreOf would provide the same result semantically. Therefore, in case of a single element in a Structured Data Information Element, the default "undefined" semantic SHOULD be used.

6. Template Management

This section introduces some more specific Template management and Template Withdrawal Message-related specifications compared to the IPFIX protocol specification [RFC5101].

First of all, the Template ID uniqueness is unchanged compared to [RFC5101]; the uniqueness is local to the Transport Session and Observation Domain that generated the Template ID. In other words, the Set ID used to export the Template Record does not influence the Template ID uniqueness.

While [RFC5101] mentions that "if an Information Element is required more than once in a Template, the different occurrences of this Information Element SHOULD follow the logical order of their treatments by the Metering Process", this rule MAY be ignored within Structured Data Information Elements.

As specified in [RFC5101], Templates that are not used anymore SHOULD be deleted. Deleting a Template implies that it MUST NOT be used within subTemplateList and subTemplateMultiList anymore. Before reusing a Template ID, the Template MUST be deleted. In order to delete an allocated Template, the Template is withdrawn through the use of a Template Withdrawal Message.

7. The Collecting Process's Side

This section introduces some more specific specifications to the Collection Process compared to Section 9 in the IPFIX protocol [RFC5101].

As opposed to the IPFIX specification in [RFC5101], IPFIX Messages with IPFIX Structured Data Information Elements change the IPFIX

concept from the Collector's point of view as the data types are present in the Data Records rather than in the Template Records. For example, a basicList Information Element in a Template Record doesn't specify the list element data type; this information is contained in the Data Record. For example, in case of a subTemplateMultiList, the Collecting Process must refer to the included Template Records in the middle of the Data Record decode.

As described in [RFC5101], a Collecting Process MUST note the Information Element identifier of any Information Element that it does not understand and MAY discard that Information Element from the Flow Record. Therefore, a Collection Process that does not support the extension specified in this document can ignore the Structured Data Information Elements in a Data Record, or it can ignore Data Records containing these new Structured Data Information Elements while continuing to process other Data Records.

If the structured data contains the "undefined" structured data type semantic, the Collecting Process MAY attempt to draw its own conclusion in terms of the semantic contained in the Data Record.

8. Defining New Information Elements Based on the New Abstract Data Types

This document specifies three new abstract data types: basicList, subTemplateList, and subTemplateMultiList. As specified in [RFC5102], the specification of new IPFIX Information Elements uses the Template specified in Section 2.1 of [RFC5102]. This Template mentioned existing and future the data types: "One of the types listed in Section 3.1 of this document or in a future extension of the information model". So new Information Elements can be specified based on the three new abstract data types.

The authors anticipate the creation of both enterprise-specific and IANA Information Elements based on the IPFIX structured data types. For example, bgpPathList, bgpSequenceList, and bgpSetList, of abstract types and semantics basicList/ordered, basicList/ordered, and basicList/exactlyOneOf respectively, would define the complete semantic of the list. This specification doesn't specify any new Information Elements beyond the ones in Section 4.3.

9. Structured Data Encoding Examples

The following examples are created solely for the purpose of illustrating how the extensions proposed in this document are encoded.

9.1. Encoding a Multicast Data Record with basicList

Consider encoding a multicast Data Record containing the following data:

Ingress If	Source IP	Destination IP	Egress Interfaces
9	192.0.2.201	233.252.0.1	1, 4, 8

Template Record for the multicast Flows, with the Template ID 256:

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																		
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
										Set ID = 2																				Length = 24 octets																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
										Template ID = 256																				Field Count = 4																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
0										ingressInterface = 10																				Field Length = 4																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
0										sourceIPv4Address = 8																				Field Length = 4																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
0										DestinationIPv4Address = 12																				Field Length = 4																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		
0										basicList = 291																				Field Length = 0xFFFF																			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-																		

Figure 11: Encoding basicList, Template Record

The list of outgoing interfaces is represented as a basicList with semantic allof, and the Length of the list is chosen to be encoded in three bytes even though it may be less than 255 octets.

The Data Set is represented as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 256           |           Length = 36           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           ingressInterface = 9           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           sourceIPv4Address = 192.0.2.201           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           DestinationIPv4Address = 233.252.0.1           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           255           |           List Length = 17           | semantic=allOf |
+-----+-----+-----+-----+-----+-----+-----+-----+
| egressInterface FieldId = 14 | egressInterface Field Length=4 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           egressInterface value 1 = 1           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           egressInterface value 2 = 4           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           egressInterface value 3 = 8           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 12: Encoding basicList, Data Record, Semantic allOf

In the example above, the basicList contains fixed-length elements. To illustrate how variable-length elements would be encoded, the same example is shown below with variable-length interface names in the basicList instead:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 256          |          Length = 44          |
+-----+-----+-----+-----+
|          ingressInterface = 9          |
+-----+-----+-----+-----+
|          sourceIPv4Address = 192.0.2.201          |
+-----+-----+-----+-----+
|          DestinationIPv4Address = 233.252.0.1          |
+-----+-----+-----+-----+
|          255          |          List Length = 25          | semantic=allOf|
+-----+-----+-----+-----+
|0| InterfaceName FieldId = 82 | InterfaceName Field Len=0xFFFF|
+-----+-----+-----+-----+
| Length = 5 |          'F'          |          'E'          |          '0'          |
+-----+-----+-----+-----+
|          '/'          |          '0'          | Length = 7 |          'F'          |
+-----+-----+-----+-----+
|          'E'          |          '1'          |          '0'          |          '/'          |
+-----+-----+-----+-----+
|          '1'          |          '0'          | Length = 5 |          'F'          |
+-----+-----+-----+-----+
|          'E'          |          '2'          |          '/'          |          '2'          |
+-----+-----+-----+-----+

```

Figure 13: Encoding basicList, Data Record with Variable-Length Elements, Semantic allOf

9.2. Encoding a Load-Balanced Data Record with a basicList

Consider encoding a load-balanced Data Record containing the following data:

Ingress If	Source IP	Destination IP	Egress Interfaces
9	192.0.2.201	233.252.0.1	1, 4, 8

So the Data Record egressed from either interface 1, 4, or 8. The Data Set is represented as follows:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 256          |          Length = 36          |
+-----+-----+-----+-----+
|          ingressInterface = 9          |
+-----+-----+-----+-----+
|          sourceIPv4Address = 192.0.2.201          |
+-----+-----+-----+-----+
|          DestinationIPv4Address = 233.252.0.1          |
+-----+-----+-----+-----+
|          255          |          List Length = 17          |sem=exactlyOne |
+-----+-----+-----+-----+
| egressInterface FieldId = 14 | egressInterface Field Length=4 |
+-----+-----+-----+-----+
|          egressInterface value 1 = 1          |
+-----+-----+-----+-----+
|          egressInterface value 2 = 4          |
+-----+-----+-----+-----+
|          egressInterface value 3 = 8          |
+-----+-----+-----+-----+

```

Note: sem=exactlyOne represents semantic=exactlyOneOf

Figure 14: Encoding basicList, Data Record, Semantic exactlyOneOf

9.3. Encoding subTemplateList

As explained in [Section 2.2](#), multiple pairs of (observationTimeMicroseconds, digestHashValue) must be collected from two different Observation Points to passively compute the one-way delay across the network. This data can be exported with an optimized Data Record that consists of the following attributes:

```

5-tuple
{ observationTimeMicroseconds 1, digestHashValue 1 }
{ observationTimeMicroseconds 2, digestHashValue 2 }
{ observationTimeMicroseconds 3, digestHashValue 3 }
{ ... , ... }

```

A subTemplateList is best suited for exporting the list of (observationTimeMicroseconds, digestHashValue). For illustration purposes, the number of elements in the list is 5; in practice, it could be more.

srcIP	dstIP	src Port	dst Port	proto	one-way delay metrics
192.0.2.1	192.0.2.105	1025	80	6	Time1, 0x0x91230613 Time2, 0x0x91230650 Time3, 0x0x91230725 Time4, 0x0x91230844 Time5, 0x0x91230978

The following Template is defined for exporting the one-way delay metrics:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
+--+																																							

Figure 15: Encoding subTemplateList, Template for One-Way Delay Metrics

The Template Record for the Optimized Data Record is as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 2           |           Length = 32 octets           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Template ID = 258    |           Field Count = 6             |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| sourceIPv4Address = 8        |           Field Length = 4           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| destinationIPv4Address = 12  |           Field Length = 4           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| sourceTransportPort = 7      |           Field Length = 2           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| destinationTransportPort= 11|           Field Length = 2           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| protocolIdentifier = 4       |           Field Length = 1           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| subTemplateList = 292       |           Field Length = 0xFFFF     |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 16: Encoding subTemplateList, Template Record

The list of (observationTimeMicroseconds, digestHashValue) is exported as a subTemplateList with semantic allOf. The Length of the subTemplateList is chosen to be encoded in three bytes even though it may be less than 255 octets.

The Data Record is represented as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 258         |           Length = 83 octets         |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           sourceIPv4Address = 192.0.2.1           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           destinationIPv4Address = 192.0.2.105    |
+-----+-----+-----+-----+-----+-----+-----+-----+
| sourceTransportPort = 1025    | destinationTransportPort = 80 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Protocol = 6 |           255           | one-way metrics list len = 63 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| semantic=allOf |           TemplateID = 257           | TimeValue1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           ... octets 2-5 of TimeValue1           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```



```

|          ... octets 6-8 of TimeValue1          |digestHashVal1=|
+-----+
|          ... 0x0x91230613          | TimeValue2  |
+-----+
|          ... octets 2-5 of TimeValue2          |
+-----+
|          ... octets 6-8 of TimeValue2          |digestHashVal2=|
+-----+
|          ... 0x0x91230650          | TimeValue3  |
+-----+
|          ... octets 2-5 of TimeValue3          |
+-----+
|          ... octets 6-8 of TimeValue3          |digestHashVal3=|
+-----+
|          ... 0x0x91230725          | TimeValue4  |
+-----+
|          ... octets 2-5 of TimeValue4          |
+-----+
|          ... octets 6-8 of TimeValue4          |digestHashVal4=|
+-----+
|          ... 0x0x91230844          | TimeValue5  |
+-----+
|          ... octets 2-5 of TimeValue5          |
+-----+
|          ... octets 6-8 of TimeValue5          |digestHashVal5=|
+-----+
|          ... 0x0x91230978          |
+-----+

```

Figure 17: Encoding subTemplateList, Data Set

9.4. Encoding subTemplateMultiList

As explained in [Section 4.5.3](#), a subTemplateMultiList is used to export a list of mixed-type content where each top-level element corresponds to a different Template Record.

To illustrate this, consider the Data Record with the following attributes:

```

5-tuple (Flow Keys), octetCount, packetCount
  attributes for filtering
    selectorId,
    selectorAlgorithm
  attributes for sampling
    selectorId,
    selectorAlgorithm,
    samplingPacketInterval,
    samplingPacketSpace

```

This example demonstrates that the Selector Report Interpretation [RFC5476] can be encoded with the subTemplateMultiList. More specifically, the example describes Property Match Filtering Selector Report Interpretation [RFC5476] used for filtering purposes, and the Systemic Count-Based Sampling as described in Section 6.5.2.1 of [RFC5476]. Some traffic will be filtered according to match properties configured, some will be sampled, some will be filtered and sampled, and some will not be filtered or sampled.

A subTemplateMultiList is best suited for exporting this variable data. A Template is defined for filtering attributes and another Template is defined for sampling attributes. A Data Record can contain data corresponding to either of the Templates, both of them, or neither of them.

Consider the example below where the following Data Record contains both filtering and sampling attributes.

Key attributes of the Data Record:

```

-----
srcIP      | dstIP      | src  | dst  | proto | octetCount | packet
            |            | Port | Port |       |            | Count
-----
2001:DB8::1 2001:DB8::2 1025   80    6      108000      120
-----

```

Filtering attributes:

```

-----
selectorId | selectorAlgorithm
-----
100        | 5 (Property Match Filtering)
-----

```

Sampling attributes:

For Systemic Count-Based Sampling as defined in [Section 6.5.2.1 of \[RFC5476\]](#) the required algorithm-specific Information Elements are:

samplingPacketInterval: number of packets selected in a row
 samplingPacketSpace: number of packets between selections

Example of a simple 1-out-of-100 systematic count-based Selector definition, where the samplingPacketInterval is 1 and the samplingPacketSpace is 99.

selectorId	selectorAlgorithm	sampling Packet Interval	sampling Packet Space
15	1 (Count-Based Sampling)	1	99

To represent the Data Record, the following Template Records are defined:

Template for filtering attributes: 259
 Template for sampling attributes: 260
 Template for Flow Record: 261

```
Flow record (261)
| (sourceIPv6Address)
| (destinationIPv6Address)
| (sourceTransportPort)
| (destinationTransportPort)
| (protocolIdentifier)
| (octetTotalCount)
| (packetTotalCount)
|
+----- filtering attributes (259)
| (selectorId)
| (selectorAlgorithm)
|
+----- sampling attributes (260)
| (selectorId)
| (selectorAlgorithm)
| (samplingPacketInterval)
| (samplingPacketSpace)
```

The following Template Record is defined for filtering attributes:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 2           |           Length = 16           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Template ID = 259    |           Field Count = 2       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| selectorId = 302             |           Field Length = 4       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| selectorAlgorithm = 304      |           Field Length = 1       |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 18: Encoding subTemplateMultiList, Template for Filtering Attributes

The Template for sampling attributes is defined as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 2           |           Length = 24           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Template ID = 260    |           Field Count = 4       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| selectorId = 302             |           Field Length = 4       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| selectorAlgorithm = 304      |           Field Length = 1       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| samplingPacketInterval = 305|           Field Length = 1       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| samplingPacketSpace = 306   |           Field Length = 1       |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 19: Encoding subTemplateMultiList, Template for Sampling Attributes

Note that while selectorAlgorithm is defined as unsigned16, and samplingPacketInterval and samplingPacketSpace are defined as unsigned32, they are compressed down to 1 octet here as allowed by Reduced Size Encoding in [Section 6.2](#) of the IPFIX protocol specifications [[RFC5101](#)].

Template for the Flow Record is defined as shown below:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 2           |           Length = 40           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Template ID = 261    |           Field Count = 8       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| sourceIPv6Address = 27       |           Field Length = 16     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| destinationIPv6Address = 28  |           Field Length = 16     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| sourceTransportPort = 7      |           Field Length = 2     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| destinationTransportPort=11  |           Field Length = 2     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| protocolIdentifier = 4       |           Field Length = 1     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0|  octetTotalCount = 85        |           Field Length = 4     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0|  packetTotalCount = 86       |           Field Length = 4     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|0| subTemplateMultiList = 293   |           Field Length = 0xFFFF |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 20: Encoding subTemplateMultiList, Template for Flow Record

A subTemplateMultiList with semantic allOf is used to export the filtering and sampling attributes. The Length field of the subTemplateMultiList is chosen to be encoded in three bytes even though it may be less than 255 octets.

The Data Record is encoded as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 261         |           Length = 73       |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           sourceIPv6Address =   ...           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           ...                   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           ...                   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           2001:DB8::1          |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

```

|               destinationIPv6Address =           ...               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|               ...               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|               ...               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|               2001:DB8::2               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| sourceTransportPort = 1025 | destinationTransportPort = 80 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| protocol = 6 |               octetTotalCount = 108000               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ... |               packetTotalCount = 120               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ... |               255               | Attributes List Length = 21 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| semantic=allOf | Filtering Template ID = 259 | Filtering Attr |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ...Length = 9 |               selectorId = ...               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ... 100 | selectorAlg = 5 | Sampling Template ID = 260 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Sampling Attributes Length=11 |               selectorId = ...       |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ... 15 | selectorAlg = 1 | Interval = 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Space = 99 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 21: Encoding subTemplateMultiList, Data Set

9.5. Encoding an Options Template Set Using Structured Data

As described in [Section 5.3](#), consider a mediation function that must aggregate Data Records from different Observation Points.

Say Observation Point 1 consists of one or more interfaces, Observation Points 2 and 3 consist of one or more linecards, and Observation Point 4 consists of one or more interfaces and one or more linecards. Without structured data, a Template would have to be defined for every possible combination to interpret the data corresponding to each of the Observation Points. However, with structured data, a basicList can be used to encode the list of interfaces and another basicList can be used to encode the list of linecards.

For the sake of simplicity, each Observation Point shown below has the IP address corresponding to the Router and an <interface> or <linecard> or <linecard and interface>. This can very well be extended to include a list of interfaces and a list of linecards using basicLists as explained above.

```
Observation Point 1: Router 1, (interface 1)
Observation Point 2: Router 2, (linecard A)
Observation Point 3: Router 3, (linecard B)
Observation Point 4: Router 4, (linecard C, interface 2)
```

The mediation function wishes to express this as a single Observation Point, in order to encode the PSAMP Selection Sequence Report Interpretation (SSRI). Recall from [RFC5476] that the PSAMP Selection Sequence Report Interpretation consists of the following fields:

```
Scope:      selectionSequenceId
Non-Scope:  one Information Element mapping the
            Observation Point
            selectorId (one or more)
```

For example, the Observation Point detailed above may be encoded in a PSAMP Selection Sequence Report Interpretation as shown below:

```
Selection Sequence 7 (Filter->Sampling):
Observation Point: subTemplateMultiList.
Router 1 (IP address = 192.0.2.11), (interface 1)
Router 2 (IP address = 192.0.2.12), (linecard A)
Router 3 (IP address = 192.0.2.13), (linecard B)
Router 4 (IP address = 192.0.2.14), (linecard C, interface 2)
selectorId: 5 (Filter, match IPv4SourceAddress 192.0.2.1)
selectorId: 10 (Sampler, Random 1 out-of ten)
```

The following Templates are defined to represent the PSAMP SSRI:

```
Template for representing PSAMP SSRI: 262
Template for representing interface: 263
Template for representing linecard: 264
Template for representing linecard and interface: 265
```

```

PSAMP SSRI (262)
| (SelectionSequenceId)
|
+--- Observation Point 1 (263)
|   (exporterIPv4Address)
|   (Interface Id)
|
+--- Observation Point 2 and 3 (264)
|   (exporterIPv4Address)
|   (linecard)
|
+--- Observation Point 4 (265)
|   (exporterIPv4Address)
|   (linecard)
|   (Interface Id)
|
| (selectorId 1)
| (selectorId 2)

```

Note that the example could further be improved with a basicList of selectorId if many Selector IDs have to be reported.

Figure 22: PSAMP SSRI to Be Encoded

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Set ID = 3           |           Length = 26           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Template ID = 262     |           Field Count = 4       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Scope Field Count = 1   | 0 | selectionSequenceId = 301 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Scope 1 Length = 4     | 0 | subTemplateMultiList = 293 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Field Length = 0xFFFF   | 0 | selectorId = 302         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Field Length = 4       | 0 | selectorId = 302         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Field Length = 4       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 23: Options Template Record for PSAMP SSRI Using subTemplateMultiList

A subTemplateMultiList with semantic allOf is used to encode the list of Observation Points.


```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 16          |
+-----+-----+-----+-----+
|          Template ID = 263   |          Field Count = 2      |
+-----+-----+-----+-----+
|0|    exporterIPv4Address = 8  |          Field Length = 4    |
+-----+-----+-----+-----+
|0|    ingressInterface = 10    |          Field Length = 4    |
+-----+-----+-----+-----+

```

Figure 24: PSAMP SSRI, Template Record for interface

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 16          |
+-----+-----+-----+-----+
|          Template ID = 264   |          Field Count = 2      |
+-----+-----+-----+-----+
|0|    exporterIPv4Address = 8  |          Field Length = 4    |
+-----+-----+-----+-----+
|0|          lineCardId = 141   |          Field Length = 4    |
+-----+-----+-----+-----+

```

Figure 25: PSAMP SSRI, Template Record for linecard

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 20          |
+-----+-----+-----+-----+
|          Template ID = 265   |          Field Count = 3      |
+-----+-----+-----+-----+
|0|    exporterIPv4Address = 8  |          Field Length = 4    |
+-----+-----+-----+-----+
|0|          lineCardId = 141   |          Field Length = 4    |
+-----+-----+-----+-----+
|0|    ingressInterface = 10    |          Field Length = 4    |
+-----+-----+-----+-----+

```

Figure 26: PSAMP SSRI, Template Record for linecard and interface

The PSAMP SSRI Data Set is represented as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Set ID = 262           |           Length = 68           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           selectionSequenceId = 7           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           255           | Observation Point List Len=49 |semantic=allOf |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP1 Template ID = 263           |           OP1 Length = 12           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Router 1 exporterIPv4Address = 192.0.2.11           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP1 ingressInterface = 1           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP2&OP3 Template ID = 264           |           OP2 & OP3 Length = 20           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Router 2 exporterIPv4Address = 192.0.2.12           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP2 lineCardId = A           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Router 3 exporterIPv4Address = 192.0.2.13           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP3 lineCardId = B           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP4 Template ID = 265           |           OP4 Length = 16           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Router 4 exporterIPv4Address = 192.0.2.14           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP4 lineCardId = C           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           OP4 ingressInterface = 2           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           selectorId = 5           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           selectorId = 10           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 27: Example of a PSAMP SSRI Data Record, Encoded Using a subTemplateMultiList

Note that the Data Record above contains multiple instances of Template 264 to represent Observation Point 2 (Router2, linecard A) and Observation Point 3 (Router3, linecard B). Instead, if a single Observation Point had both linecard A and linecard B, a basicList would be used to represent the list of linecards.

10. Relationship with the Other IPFIX Documents

10.1. Relationship with Reducing Redundancy

"Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports" [RFC5473] describes a bandwidth saving method for exporting Flow or packet information using the IP Flow Information Export (IPFIX) protocol.

It defines the commonPropertiesID Information Element for exporting Common Properties.

10.1.1. Encoding Structured Data Element Using Common Properties

When Structured Data Information Elements contain repeated elements, these elements may be replaced with a commonPropertiesID Information Element as specified in [RFC5473]. The replaced elements may include the basicList, subTemplateList, and subTemplateMultiList Information Elements.

This technique might help reducing the bandwidth requirements for the export. However, a detailed analysis of the gain has not been done; refer to Section 8.3 of [RFC5473] for further considerations.

10.1.2. Encoding Common Properties Elements with Structured Data Information Element

Structured Data Information Element MAY be used to define a list of commonPropertiesID, as a replacement for the specifications in [RFC5473].

Indeed, the example in Figures 1 and 2 of [RFC5473] can be encoded with the specifications in this document.

sourceAddressA	sourcePortA	<Flow1 information>
sourceAddressA	sourcePortA	<Flow2 information>
sourceAddressA	sourcePortA	<Flow3 information>
sourceAddressA	sourcePortA	<Flow4 information>
...

Figure 28: Common and Specific Properties Exported Together
[RFC5473]

index for properties A	sourceAddressA	sourcePortA
...
index for properties A	<Flow1 information>	
index for properties A	<Flow2 information>	
index for properties A	<Flow3 information>	
index for properties A	<Flow4 information>	

Figure 29: Common and Specific Properties Exported Separately
According to [RFC5473]

sourceAddressA	sourcePortA	<Flow1 information>
		<Flow2 information>
		<Flow3 information>
		<Flow4 information>
		...

Figure 30: Common and Specific Properties Exported with
Structured Data Information Element

The example in Figure 28 could be encoded with a basicList if the <Flow information> represents a single Information Element, with a subTemplateList if the <Flow information> represents a Template Record, or with a subTemplateMultiList if the <Flow information> is composed of different Template Records.

Using Structured Data Information Elements as a replacement for the techniques specified in "Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports" [RFC5473] offers the advantage that a single Template Record is defined. Hence, the Collector's job is simplified in terms of Template management and combining Template/Options Template Records.

However, it must be noted that using Structured Data Information Elements as a replacement for the techniques specified in "Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports" only applies to simplified cases. For example, the "Multiple Data Reduction" ([Section 7.1 \[RFC5473\]](#)) might be too complex to encode with Structured Data Information Elements.

10.2. Relationship with Guidelines for IPFIX Testing

[RFC5471] presents a list of tests for implementers of IP Flow Information Export (IPFIX) compliant Exporting Processes and Collecting Processes.

Although [RFC5471] doesn't define any structured data element specific tests, the Structured Data Information Elements can be used in many of the [RFC5471] tests.

The [RFC5471] series of test could be useful because the document specifies that every Information Element type should be tested. However, not all cases from this document are tested in [RFC5471].

The following sections are especially noteworthy:

3.2.1. Transmission of Template with Fixed-Size Information Elements

- each data type should be used in at least one test. The new data types specified in [Section 4.1](#) should be included in this test.

3.2.2. Transmission of Template with Variable-Length Information Elements

- this test should be expanded to include Data Records containing variable length basicList, subTemplateList, and subTemplateMultiList Information Elements.

3.3.1. Enterprise-Specific Information Elements

- this test should include the export of basicList, subTemplateList, and subTemplateMultiList Information Elements containing Enterprise-specific Information Elements, e.g., see the example in Figure 2.

3.3.3. Multiple Instances of the Same Information Element in One Template

- this test should verify that multiple instances of the basicList, subTemplateList, and subTemplateMultiList Information Elements are accepted.

3.5. Stress/Load Tests

- since the structured data types defined here allow modeling of complex data structures, they may be useful for stress testing both Exporting Processes and Collecting Processes.

10.3. Relationship with IPFIX Mediation Function

The Structured Data Information Elements would be beneficial for the export of aggregated Data Records in mediation function, as was demonstrated with the example of the aggregated Observation Point in [Section 5.3](#).

11. IANA Considerations

This document specifies several new IPFIX abstract data types, a new IPFIX Data Type Semantic, and several new Information Elements.

Two new IPFIX registries have been created, and the existing IPFIX Information Element registry has been updated as detailed below.

11.1. New Abstract Data Types

[Section 4.1](#) of this document specifies several new IPFIX abstract data types. Per [Section 6](#) of the IPFIX information model [[RFC5102](#)], new abstract data types can be added to the IPFIX information model in the IPFIX Information Element Data Types registry.

Abstract data types that have been added to the IPFIX Information Element Data Types registry are listed below.

11.1.1. basicList

The type "basicList" represents a list of any Information Element used for single-valued data types.

11.1.2. subTemplateList

The type "subTemplateList" represents a list of a structured data type, where the data type of each list element is the same and corresponds with a single Template Record.

11.1.3. subTemplateMultiList

The type "subTemplateMultiList" represents a list of structured data types, where the data types of the list elements can be different and correspond with different Template definitions.

11.2. New Data Type Semantics

Section 4.2 of this document specifies a new IPFIX Data Type Semantic. Per Section 3.2 of the IPFIX information model [RFC5102], new data type semantics can be added to the IPFIX information model. Therefore, the IANA IPFIX informationElementSemantics registry [IANA-IPFIX], which contains all the data type semantics from Section 3.2 of [RFC5102], has been augmented with the "list" value below.

11.2.1. list

A list is a structured data type, being composed of a sequence of elements, e.g., Information Element, Template Record.

11.3. New Information Elements

Section 4.3 of this document specifies several new Information Elements that have been created in the IPFIX Information Element registry [IANA-IPFIX].

New Information Elements that have been added to the IPFIX Information Element registry are listed below.

11.3.1. basicList

Name: basicList

Description:

Specifies a generic Information Element with a basicList abstract data type. Examples include a list of port numbers, and a list of interface indexes.

Abstract Data Type: basicList

Data Type Semantics: list

ElementId: 291

Status: current

11.3.2. subTemplateList

Name: subTemplateList
Description:
Specifies a generic Information Element with a subTemplateList abstract data type.
Abstract Data Type: subTemplateList
Data Type Semantics: list
ElementId: 292
Status: current

11.3.3. subTemplateMultiList

Name: subTemplateMultiList
Description:
Specifies a generic Information Element with a subTemplateMultiList abstract data type.
Abstract Data Type: subTemplateMultiList
Data Type Semantics: list
ElementId: 293
Status: current

11.4. New Structured Data Semantics

[Section 4.4](#) of this document specifies a series of new IPFIX structured data type semantics, which is expressed as an 8-bit value. This requires the creation of a new "IPFIX Structured Data Types Semantics" IPFIX subregistry [[IANA-IPFIX](#)].

Entries may be added to this subregistry subject to a Standards Action [[RFC5226](#)]. Initially, this registry includes all the structured data type semantics listed below.

11.4.1. undefined

Name: undefined

Description: The "undefined" structured data type semantic specifies that the semantic of list elements is not specified and that, if a semantic exists, then it is up to the Collecting Process to draw its own conclusions. The "undefined" structured data type semantic is the default structured data type semantic.

Value: 0xFF

Reference: [RFC 6313](#)

11.4.2. noneOf

Name: noneOf

Description: The "noneOf" structured data type semantic specifies that none of the elements are actual properties of the Data Record.

Value: 0x00

Reference: [RFC 6313](#)

11.4.3. exactlyOneOf

Name: exactlyOneOf

Description: The "exactlyOneOf" structured data type semantic specifies that only a single element from the structured data is an actual property of the Data Record. This is equivalent to a logical XOR operation.

Value: 0x01

Reference: [RFC 6313](#)

11.4.4. oneOrMoreOf

Name: oneOrMoreOf

Description: The "oneOrMoreOf" structured data type semantic specifies that one or more elements from the list in the structured data are actual properties of the Data Record. This is equivalent to a logical OR operation.

Value: 0x02

Reference: [RFC 6313](#)

11.4.5. allOf

Name: allOf

Description: The "allOf" structured data type semantic specifies that all of the list elements from the structured data are actual properties of the Data Record.

Value: 0x03

Reference: [RFC 6313](#)

11.4.6. ordered

Name: ordered Description: The "ordered" structured data type semantic specifies that elements from the list in the structured data are ordered.

Value: 0x04

Reference: [RFC 6313](#)

12. Security Considerations

The addition of complex data types necessarily complicates the implementation of the Collector. This could easily result in new security vulnerabilities (e.g., buffer overflows); this creates additional risk in cases where either Datagram Transport Layer Security (DTLS) is not used or if the Observation Point and Collector belong to different trust domains. Otherwise, the same security considerations as for the IPFIX protocol [[RFC5101](#)] and the IPFIX information model [[RFC5102](#)] apply.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC5101] Claise, B., Ed., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", [RFC 5101](#), January 2008.
- [RFC5102] Quittek, J., Bryant, S., Claise, B., Aitken, P., and J. Meyer, "Information Model for IP Flow Information Export", [RFC 5102](#), January 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), May 2008.

13.2. Informative References

- [RFC3917] Quittek, J., Zseby, T., Claise, B., and S. Zander, "Requirements for IP Flow Information Export (IPFIX)", [RFC 3917](#), October 2004.

- [RFC5103] Trammell, B. and E. Boschi, "Bidirectional Flow Export Using IP Flow Information Export (IPFIX)", [RFC 5103](#), January 2008.
- [RFC5470] Sadasivan, G., Brownlee, N., Claise, B., and J. Quittek, "Architecture for IP Flow Information Export", [RFC 5470](#), March 2009.
- [RFC5471] Schmoll, C., Aitken, P., and B. Claise, "Guidelines for IP Flow Information Export (IPFIX) Testing", [RFC 5471](#), March 2009.
- [RFC5472] Zseby, T., Boschi, E., Brownlee, N., and B. Claise, "IP Flow Information Export (IPFIX) Applicability", [RFC 5472](#), March 2009.
- [RFC5473] Boschi, E., Mark, L., and B. Claise, "Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports", [RFC 5473](#), March 2009.
- [RFC5475] Zseby, T., Molina, M., Duffield, N., Niccolini, S., and F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection", [RFC 5475](#), March 2009.
- [RFC5476] Claise, B., Ed., Johnson, A., and J. Quittek, "Packet Sampling (PSAMP) Protocol Specifications", [RFC 5476](#), March 2009.
- [RFC5477] Dietz, T., Claise, B., Aitken, P., Dressler, F., and G. Carle, "Information Model for Packet Sampling Exports", [RFC 5477](#), March 2009.
- [IANA-IPFIX] IANA, "IP Flow Information Export (IPFIX) Entities", [<http://www.iana.org/>](http://www.iana.org/).

14. Acknowledgements

The authors would like to thank Zhipu Jin, Nagaraj Varadharajan, Brian Trammel, Atsushi Kobayashi, and Rahul Patel for their feedback, and Gerhard Muenz, for proofreading the document.

Appendix A. Additions to XML Specification of IPFIX Information Elements and Abstract Data Types

This appendix contains additions to the machine-readable description of the IPFIX information model coded in XML in Appendices A and B in [RFC5102]. Note that this appendix is of informational nature, while the text in [Section 4](#) (generated from this appendix) is normative.

The following field definitions are appended to the IPFIX information model in [Appendix A of \[RFC5102\]](#).

```
<field name="basicList"
      dataType="basicList"
      group="structured-data"
      dataTypeSemantics="List"
      elementId="291" applicability="all" status="current">
  <description>
    <paragraph>
      Represents a list of zero or more instances of
      any Information Element, primarily used for
      single-valued data types. Examples include a list of port
      numbers, list of interface indexes, and a list of AS in a
      BGP AS-PATH.
    </paragraph>
  </description>
</field>

<field name="subTemplateList"
      dataType="subTemplateList"
      group="structured-data"
      dataTypeSemantics="List"
      elementId="292" applicability="all" status="current">
  <description>
    <paragraph>
      Represents a list of zero or more instances of a
      structured data type, where the data type of each list
      element is the same and corresponds with a single
      Template Record. Examples include a structured data type
      composed of multiple pairs of ("MPLS label stack entry
      position", "MPLS label stack value"), a structured data
      type composed of performance metrics, and a structured data
      type composed of multiple pairs of IP address.
    </paragraph>
  </description>
</field>
```

```
<field name="subTemplateMultiList"
  dataType="subTemplateMultiList"
  group="structured-data"
  dataTypeSemantics="List"
  elementId="293" applicability="all" status="current">
  <description>
    <paragraph>
      Represents a list of zero or more instances of
      structured data types, where the data type of each list
      element can be different and corresponds with
      different Template definitions. Examples include, a
      structured data type composed of multiple access-list
      entries, where entries can be composed of different
      criteria types.
    </paragraph>
  </description>
</field>
```

The following structured data type semantic definitions are appended to the IPFIX information model in [Appendix A of \[RFC5102\]](#).

```
<structuredDataTypeSemantics>
  <structuredDataTypeSemantic name="undefined" value="255">
    <description>
      <paragraph>
        The "undefined" structured data type semantic specifies
        that the semantic of list elements is not specified and
        that, if a semantic exists, then it is up to the
        Collecting Process to draw its own conclusions. The
        "undefined" structured data type semantic is the default
        structured data type semantic.
      </paragraph>
    </description>
  </structuredDataTypeSemantic>

  <structuredDataTypeSemantic name="noneOf" value="0">
    <description>
      <paragraph>
        The "noneOf" structured data type semantic specifies
        that none of the elements are actual properties of the
        Data Record.
      </paragraph>
    </description>
  </structuredDataTypeSemantic>
```

```
<structuredDataTypeSemantic name="exactlyOneOf" value="1">
  <description>
    <paragraph>
      The "exactlyOneOf" structured data type semantic
      specifies that only a single element from the structured
      data is an actual property of the Data Record. This is
      equivalent to a logical XOR operation.
    </paragraph>
  </description>
</structuredDataTypeSemantic>

<structuredDataTypeSemantic name="oneOrMoreOf" value="2">
  <description>
    <paragraph>
      The "oneOrMoreOf" structured data type semantic
      specifies that one or more elements from the list in the
      structured data are actual properties of the Data
      Record. This is equivalent to a logical OR operation.
    </paragraph>
  </description>
</structuredDataTypeSemantic>

<structuredDataTypeSemantic name="allOf" value="3">
  <description>
    <paragraph>
      The "allOf" structured data type semantic specifies that
      all of the list elements from the structured data are
      actual properties of the Data Record.
    </paragraph>
  </description>
</structuredDataTypeSemantic>

<structuredDataTypeSemantic name="ordered" value="4">
  <description>
    <paragraph>
      The "ordered" structured data type semantic specifies
      that elements from the list in the structured data are
      ordered.
    </paragraph>
  </description>
</structuredDataTypeSemantic>
</structuredDataTypeSemantics>
```

The following schema definitions are appended to the abstract data types defined in [Appendix B of \[RFC5102\]](#). This schema and its namespace are registered by IANA at <http://www.iana.org/assignments/xml-registry/schema/ipfix.xsd>.

```
<simpleType name="dataType">
  <restriction base="string">
    <enumeration value="basicList">
      <annotation>
        <documentation>
          Represents a list of zero or more instances of
          any Information Element, primarily used for
          single-valued data types. Examples include a list of port
          numbers, a list of interface indexes, and a list of AS in a
          BGP AS-PATH.
        </documentation>
      </annotation>
    </enumeration>
    <enumeration value="subTemplateList">
      <annotation>
        <documentation>
          Represents a list of zero or more instances of a
          structured data type, where the data type of each list
          element is the same and corresponds with a single
          Template Record. Examples include a structured data type
          composed of multiple pairs of ("MPLS label stack entry
          position", "MPLS label stack value"), a structured
          data type composed of performance metrics, and a
          structured data type composed of multiple pairs of IP
          address.
        </documentation>
      </annotation>
    </enumeration>
    <enumeration value="subTemplateMultiList">
      <annotation>
        <documentation>
          Represents a list of zero or more instances of
          structured data types, where the data type of each
          list element can be different and corresponds with
          different Template definitions. An example is a
          structured data type composed of multiple
          access-list entries, where entries can be
          composed of different criteria types.
        </documentation>
      </annotation>
    </enumeration>
  </restriction>
</simpleType>
```

```
<simpleType name="dataTypeSemantics">
  <restriction base="string">
    <enumeration value="List">
      <annotation>
        <documentation>
          Represents an arbitrary-length sequence of structured
          data elements, either composed of regular Information
          Elements or composed of data conforming to a Template
          Record.
        </documentation>
      </annotation>
    </enumeration>
  </restriction>
</simpleType>

<complexType name="structuredDataTypeSemantics">
  <sequence>
    <element name="structuredDataTypeSemantic"
      minOccurs="1" maxOccurs="unbounded">
      <complexType>
        <sequence>
          <element name="description" type="text"/>
        </sequence>
        <attribute name="name" type="string" use="required"/>
        <attribute name="value" type="unsignedByte" use="required"/>
      </complexType>
    </element>
  </sequence>
</complexType>

<element name="structuredDataTypeSemantics"
  type="structuredDataTypeSemantics">
  <annotation>
    <documentation>
      Structured data type semantics express the relationship
      among multiple list elements in a structured data
      Information Element.
    </documentation>
  </annotation>
</element>
```


Appendix B. Encoding IPS Alert Using Structured Data Information Elements

In this section, an IPS alert example is used to demonstrate how complex data and multiple levels of hierarchy can be encoded using Structured Data Information Elements. Also, this example demonstrates how a `basicList` of `subTemplateLists` can be used to represent semantics at multiple levels in the hierarchy.

An IPS alert consists of the following mandatory attributes: `signatureId`, `protocolIdentifier`, and `riskRating`. It can also contain zero or more participants, and each participant can contain zero or more attackers and zero or more targets. An attacker contains the attributes `sourceIPv4Address` and `applicationId`, and a target contains the attributes `destinationIPv4Address` and `applicationId`.

Note that the `signatureId` and `riskRating` Information Element fields are created for these examples only; the Field IDs are shown as N/A. The `signatureId` helps to uniquely identify the IPS signature that triggered the alert. The `riskRating` identifies the potential risk, on a scale of 0-100 (100 being most serious), of the traffic that triggered the alert.

Consider the example described in case study 2 of [Section 5.6](#). The IPS alert contains participants encoded as a `subTemplateList` with semantic `allof`. Each participant uses a `basicList` of `subTemplateLists` to represent attackers and targets. For the sake of simplicity, the alert has two participants P1 and P2. In participant P1, attacker A1 or A2 attacks target T1. In participant P2, attacker A3 attacks targets T2 and T3.

Participant P1:

```
(basicList, allof,
  (subTemplateList, exactlyOneOf, attacker A1, A2)
  (subTemplateList, undefined, target T1)
)
```

Participant P2:

```
(basicList, allof,
  (subTemplateList, undefined, attacker A3,
  (subTemplateList, allof, targets T2, T3)
)
```

Alert :

```
(subTemplateList, allof, Participant P1, Participant P2)
```

sigId	protocol Id	risk Rating	participant			
			attacker		target	
			IP	appId	IP	appId
1003	17	10	192.0.2.3	103	192.0.2.103	3001
			192.0.2.4	104		
			192.0.2.5	105	192.0.2.104	4001
					192.0.2.105	5001

Participant P1 contains:

Attacker A1: (IP, appId)=(192.0.2.3, 103)

Attacker A2: (IP, appId)=(192.0.2.4, 104)

Target T1: (IP, appId)= (192.0.2.103, 3001)

Participant P2 contains:

Attacker A3: (IP, appId) = (192.0.2.5, 105)

Target T2: (IP, appId)= (192.0.2.104, 4001)

Target T3: (IP, appId)= (192.0.2.105, 5001)

To represent an alert, the following Templates are defined:

Template for target (268)

Template for attacker (269)

Template for participant (270)

Template for alert (271)

```

alert (271)
| (signatureId)
| (protocolIdentifier)
| (riskRating)
|
+----- participant (270)
|
| +----- attacker (269)
| | (sourceIPv4Address)
| | (applicationId)
| |
| +----- target (268)
| | (destinationIPv4Address)
| | (applicationId)

```

Note that the attackers are always composed of a single applicationId, while the targets typically have multiple applicationIds; for the sake of simplicity, this example shows only one applicationId in the target.

Template Record for target, with the Template ID 268:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Set ID = 2										Length = 16 octets																													
Template ID = 268										Field Count = 2																													
0 destinationIPv4Address = 12										Field Length = 4																													
0 applicationId = 95										Field Length = 4																													

Figure 31: Encoding IPS Alert, Template for Target

Template Record for attacker, with the Template ID 269:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 16 octets          |
+-----+-----+-----+-----+
|          Template ID = 269   |          Field Count = 2           |
+-----+-----+-----+-----+
|0|          sourceIPv4Address = 8      |          Field Length = 4      |
+-----+-----+-----+-----+
|0|          applicationId = 95        |          Field Length = 4      |
+-----+-----+-----+-----+

```

Figure 32: Encoding IPS Alert, Template for Attacker

Template Record for participant, with the Template ID 270:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 12 octets          |
+-----+-----+-----+-----+
|          Template ID = 270   |          Field Count = 1           |
+-----+-----+-----+-----+
|0|          basicList = 291      |          Field Length = 0xFFFF  |
+-----+-----+-----+-----+

```

Figure 33: Encoding IPS Alert, Template for Participant

The Template Record for the participant has one basicList Information Element, which is a list of subTemplateLists of attackers and targets.

Template Record for IPS alert, with the Template ID 271:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 2          |          Length = 24 octets          |
+-----+-----+-----+-----+
|          Template ID = 271   |          Field Count = 4           |
+-----+-----+-----+-----+
|0|      signatureId = N/A      |          Field Length = 2         |
+-----+-----+-----+-----+
|0|      protocolIdentifier = 4  |          Field Length = 1         |
+-----+-----+-----+-----+
|0|      riskRating = N/A       |          Field Length = 1         |
+-----+-----+-----+-----+
|0|      subTemplateList = 292  |          Field Length = 0xFFFF   |
+-----+-----+-----+-----+

```

Figure 34: Encoding IPS Alert, Template for IPS Alert

The subTemplateList in the alert Template Record contains a list of participants.

The Length of basicList and subTemplateList are encoded in three bytes even though they may be less than 255 octets.

The Data Set is represented as follows:

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|          Set ID = 271        |          Length = 102         |
+-----+-----+-----+-----+
|      signatureId = 1003      | protocolId=17 | riskRating=10 |
+-----+-----+-----+-----+
|      255      |participant List Length = 91 |semantic=allof |
+-----+-----+-----+-----+
| participant Template ID = 270 |      255      | P1 List Len = |
+-----+-----+-----+-----+
|      41      | semantic=allof | P1 List Field ID = 292 |
+-----+-----+-----+-----+
| P1 List Field ID Len = 0xFFFF |      255      | P1 attacker ... |
+-----+-----+-----+-----+
| List Len = 19 |sem=exactlyOne | P1 attacker Template ID = 269 |
+-----+-----+-----+-----+
|          P1 attacker A1 sourceIPv4Address = 192.0.2.3          |
+-----+-----+-----+-----+
|          P1 attacker A1 applicationId = 103                    |
+-----+-----+-----+-----+

```

```

+-----+
|           P1 attacker A2 sourceIPv4Address = 192.0.2.4           |
+-----+
|           P1 attacker A2 applicationId = 104                     |
+-----+
|      255      | P1 target List Len = 11      | sem=undefined |
+-----+
| P1 target Template ID = 268 | P1 target T1 destinationIPv4 |
+-----+
| ... Address = 192.0.2.103   | P1 target T1 applicationId = ... |
+-----+
| ...      3001      |      255      | P2 List Len = |
+-----+
| ...  41      | semantic=allOf | P2 List Field ID = 292 |
+-----+
| P2 List Field ID Len = 0xFFFF |      255      | P2 attacker ... |
+-----+
| List Len = 11 | sem=undefined | P2 attacker Template ID = 269 |
+-----+
|           P2 attacker A3 sourceIPv4Address = 192.0.2.5           |
+-----+
|           P2 attacker A3 applicationId = 105                     |
+-----+
|      255      | P2 target List Len = 19 | semantic=allOf |
+-----+
| P2 target Template ID = 268 | P2 target T2 destinationIPv4 |
+-----+
| ... Address = 192.0.2.104   | P2 target T2 applicationId = ... |
+-----+
| ...      4001      | P2 target T3 destinationIPv4 |
+-----+
| ... Address = 192.0.2.105   | P2 target T3 applicationId = ... |
+-----+
| ...      5001      | |
+-----+

```

Note: sem=exactlyOne represents semantic=exactlyOneOf

Figure 35: Encoding IPS Alert, Data Set

Authors' Addresses

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
Diegem 1813
Belgium

Phone: +32 2 704 5622
EMail: bclaise@cisco.com

Gowri Dhandapani
Cisco Systems, Inc.
13615 Dulles Technology Drive
Herndon, Virginia 20171
United States

Phone: +1 408 853 0480
EMail: gowri@cisco.com

Paul Aitken
Cisco Systems, Inc.
96 Commercial Quay
Commercial Street
Edinburgh, EH6 6LX
United Kingdom

Phone: +44 131 561 3616
EMail: paitken@cisco.com

Stan Yates
Cisco Systems, Inc.
7100-8 Kit Creek Road
PO Box 14987
Research Triangle Park, North Carolina 27709-4987
United States

Phone: +1 919 392 8044
EMail: syates@cisco.com