Python fixbuf Documentation

Release 0.2.0

Carnegie Mellon University

February 02, 2015

CONTENTS

1 `pyfixbuf` API Documentation **1**

1.1 InfoElement 1
1.2 InfoElementSpec 4
1.3 InfoModel 4
1.4 Template 5
1.5 Session 7
1.6 Exporter 8
1.7 Collector 9
1.8 Record 9
1.9 Buffer 13
1.10 STML 14
1.11 STMLEntry 16
1.12 STL 17
1.13 BL 19
1.14 Listener 20
1.15 Pre-defined Information Element Lists 21

2 Pyfixbuf Examples **29**

2.1 Collector Example 29

Index **35**

CHAPTER

ONE

PYFIXBUF API DOCUMENTATION

1.1 InfoElement

Information Elements make up the IPFIX Information Model and IPFIX templates. All Information Elements consist of a unique and meaningful name, a private enterprise number (PEN), a numeric identifier, a length, and a data type. libfixbuf adds, by default, the IANA approved Information Elements to the Information Model. IANA's Information Elements have a private enterprise number of 0. pyfixbuf groups the YAF-defined Information Elements, CERT PEN 6871, by protocol. YAF_LIST and YAF_STATS are necessary for collecting default input streams from YAF. See the tables at the bottom of this page for a list of the YAF pre-defined information elements by protocol.

If an Information Element (IE) is initialized with the ENDIAN flag set, the IE is an integer and will be endian-converted on transcode. If the REVERSIBLE flag is set, a second, reverse information element will be added to the Information Model.

If an Information Element is initialized with a DataType then the appropriate Python data type will be returned. Otherwise, the value of the Information Element retrieved will be in a Byte Array. If the Information Element is of type STRING or LIST, the IE length should be VARLEN. OCTET_ARRAYS may or may not be variable length. The following is a list of acceptable data types, which are stored as an enumeration in libfixbuf. When defining an Information Element both the type and integer value are accepted.

Type	Integer Value	Length	Python Return Type
DataType.OCTET_ARRAY	0	VARLEN	Byte Array
DataType.UINT8	1	1	Integer
DataType.UINT16	2	2	Long
DataType.UINT32	3	4	Long
DataType.UINT64	4	8	Long
DataType.INT8	5	1	Long
DataType.INT16	6	2	Long
DataType.INT32	7	4	Long
DataType.INT64	8	8	Long
DataType.FLOAT32	9	4	Float
DataType.FLOAT64	10	8	Float
DataType.BOOL	11	1	Bool
DataType.MAC_ADDR	12	6	String
DataType.STRING	13	VARLEN	String
DataType.SECONDS	14	4	Long
DataType.MILLISECONDS	15	8	Long
DataType.MICROSECONDS	16	8	Long
DataType.NANOSECONDS	17	8	Long
DataType.IP4ADDR	18	4	String
DataType.IP6ADDR	19	16	String
DataType.BASIC_LIST	20	VARLEN	BL
DataType.SUB_TMPL_LIST	21	VARLEN	STL
DataType.SUB_TMPL_MULTI_LIST	22	VARLEN	STML

Units, min, max, semantic, and description are all optional parameters to further describe an information element. If the process is exporting Information Element Type Option Records (**RFC 5610**), this information will help the collecting process identify the type of information contained in the value of an Information Element. Valid Units are listed in the table below.

Units	Integer Value
Units.NONE	0
Units.BITS	1
Units.OCTETS	2
Units.PACKETS	3
Units.FLOWS	4
Units.SECONDS	5
Units.MILLISECONDS	6
Units.MICROSECONDS	7
Units.NANOSECONDS	8
Units.WORDS	9
Units.MESSAGES	10
Units.HOPS	11
Units.ENTRIES	12

The following table lists the available Semantic values:

Semantic	Integer Value
Semantic.DEFAULT	0
Semantic.QUANTITY	1
Semantic.TOTALCOUNTER	2
Semantic.DELTACOUNTER	3
Semantic.IDENTIFIER	4
Semantic.FLAGS	5
Semantic.LIST	6

class `pyfixbuf.`**`InfoElement`**(*name : str*, *enterprise_number : int*, *id : int*[, *length=VARLEN*, *reversible=False*, *endian=False*, *type=DataType.OCTET_ARRAY*, *units=Units.NONE*, *min=0*, *max=0*, *semantic=Semantic.DEFAULT*, *description=None*])

Creates a new Information Element (IE) using the given *name*, *enterprise_number*, and *id*, and optional *length*, *reversible* flag, *endian* flags, *datatype*, *units*, *min*, *max*, *semantic*, and *description*. An Information Element identifies a type of data to be stored and transmitted via IPFIX.

If no *length* is provided, the IE is defined as having a variable length. All Strings should be variable length.

If *endian* is set, the IE is assumed to be an integer and will be converted to and from network byte order upon transcoding.

If *reversible* is set, a second IE is created for the same information in the reverse direction. [The reversed IE's name is the same *name*, but with `reverse` prepended.]

If *type* is set, pyfixbuf will know how to print values of this type. Otherwise the value of the element will be a byte array. See the above table for a list of types.

units optionally defines the units of an Information Element. See the above table for a list of units.

min optionally defines the minimum value of an Information Element.

max optionally defines the maximum value of an Information Element.

semantic optionally defines the semantics of an Information Element. See the above table for a list of semantics.

description optionally contains a human-readable description of an Information Element.

`name`

The name, a string, associated with the InfoElement.

`ent`

The Private Enterprise Number (PEN) associated with the InfoElement. Default Information Elements will have a PEN of 0. An integer with max value 2^32.

`id`

The Information Element ID that, with the PEN, uniquely identifies the Information Element. ID is an integer with max value 65535.

`length`

The length associated with the Information Element. This is the amount of memory allocated for the Information Element. If the Information Element is of variable length, length will contain the size of the fbVarfield struct.

`type`

The data type associated with the Information Element. This is stored as an enumeration in libfixbuf and can have values 0-23. If type is not defined, the default type is 0, DataType.OCTET_ARRAY. If the Information Element is defined as VARLEN, the default type is 14, DataType.STRING.

`units`

The units associated with the Information Eleemnt. This is stored as an enumeration in libfixbuf and can have values 0-13. If units are not defined, the default is Units.NONE.

`min`

If a range is defined with the Information Element, min is the mininum value accepted. Valid values are 0 - 2^64.

`max`

If a range is defined for an Information Element, max is the maximum value accepted. Valid values are 0 - 2^64.

semantic
Semantic value for an Information Element. This is stored as an enumeration in libfixbuf and can have values 0 - 6. The default semantic is 0, Semantic.DEFAULT.

description
Description of an Information Element. This is a string. Default is None.

reversible
True if an Information Element is defined as reversible.

endian
True if an Information Element is defined as endian.

Examples:

```
>>> foo = pyfixbuf.InfoElement('fooname', CERT_PEN, 722, units=pyfixbuf.Units.WORDS)
>>> bar = pyfixbuf.InfoElement('barname', 123, 565, 1, reversible=True, endian=True)
>>> foo2 = pyfixbuf.InfoElement('fooname2', 0, 888, 3, type=pyfixbuf.DataType.OCTET_ARRAY)
>>> flo = pyfixbuf.InfoElement('flo_element', 0, 452, 8, endian=True, type=8)
```

1.2 InfoElementSpec

Information Element Specifications (InfoElementSpec) are used to name an information element for inclusion in a template. The Information Element must have already been defined and added to the Information Model. An InfoElementSpec contains the exact name of the defined Information Element and an optional length override.

class pyfixbuf.**InfoElementSpec**(*name : str*[, *length=0*])
Creates a new Information Element Specification using the given *name*, and optional override *length*. An IPFIX Template is made up of one or more InfoElementSpec.

The given *name* must be a defined Information Element in the Information Model before adding the InfoElementSpec to a class:*Template*.

If *length* is nonzero, it will replace the default length of this Information Element (often used for reduced-length encoding).

Examples:

```
>>> spec1 = pyfixbuf.InfoElementSpec("fooname")
>>> spec2 = pyfixbuf.InfoElementSpec("sourceTransportPort")
>>> spec3 = pyfixbuf.InfoElementSpec("flo_element", 4)
```

name
The Information Element Specification name.

length
The length override for the Information Element Specification. If a length override is not specified at initialization, the value of the InfoElementSpec length will be 0, and the length used for transcode will be the length as defined in the InfoModel.

1.3 InfoModel

The InfoModel type implements an IPFIX Information Model, adding the IANA managed Information Elements by default.

class `pyfixbuf.`**`InfoModel`**

An IPFIX Information Model stores all of the Information Elements that can be collected or exported by a Collecting or Exporting Process.

The `InfoModel` constructor creates a new Information Model and adds the default IANA-managed Information Elements.

`add_element`(*element : InfoElement*)

Adds the given `InfoElement`, *element*, to the `InfoModel`.

`add_element_list`(*list: List*)

Adds a *list* of `InfoElement` to the `InfoModel`.

`get_element_length`(*name: str*[, *type: int*]) → length

Returns the default length of the Information Element with the given *name* in the Information Model. If *type* (BASICLIST, SUBTEMPLATELIST, SUBTEMPLATEMULTILIST) is given, it assumes this element will be used in a list and the length of the list structure in libfixbuf will be returned. If the Information Element is a variable length (VARLEN) element, the length that will be returned is the length of the fbVarfield_t structure in libfixbuf. To get the length of the Information Element as it is defined in the Information Model, use `get_element` to return the `InfoElement` and the length attribute.

`get_element`([*name: str*, *id: int*, *ent: int*]) → InfoElement

Returns the `InfoElement` given the *name* or *id* and *ent*.

`get_element_type`(*name: str*) → type

Returns the type of the Information Element as defined in the `InfoModel` given the Information Element *name*.

`add_options_element`(*rec : Record*)

Add the information element contained in the Options `Record`. Use this method for incoming Options Records that contain Information Element Type Information.

Examples:

```
>>> model = pyfixbuf.InfoModel()
>>> model.add_element(foo);
>>> model.add_element_list([foo, bar, flo])
>>> model.add_element_list(pyfixbuf.YAF_DNS_LIST) # adds all YAF DNS DPI elements
>>> length = model.get_element_length("sourceTransportPort")
>>> print length
2
```

1.4 Template

The `Template` type implements an IPFIX Template or an IPFIX Options Template. IPFIX templates contain one or more Information Elements. If a certain sequence of elements is desired, each Information Element (`InfoElementSpec`) must be added to the template in the desired order. Templates are stored by Template ID and type (internal, external) per domain in a `Session`. Template IDs of data sets are numbered from 256 to 65535. Templates are given a template ID when they are added to a `Session`. The only difference between Data Templates and Options Templates is that Options Templates have a scope associated with them, which gives the context of reported Information Elements in the Data Records.

An Internal Template is how fixbuf decides what the data should look like when it is transcoded. For this reason, an internal template should match the corresponding `Record`, in terms of the order of Information Elements. An External Template is sent before the exported data so that the Collecting Process is able to process IPFIX messages without necessarily knowing the interpretation of all data records.

class `pyfixbuf.`**`Template`**(*InfoModel*)

Creates a new Template using the given *model*. An IPFIX Template is an ordered list of the Information Elements that are to be collected or exported. For export, the order of Information Elements in the Templates determines how the data will be exported.

If *type* is given, an Information Element Type Information Options Template will be created. The appropriate elements will automatically be added to the template and the scope will be sent. See **RFC 5610** for more information.

Once a Template has been added to the session, it can not be altered.

A `Template` can be accessed like a dictionary or a list to retrieve a specific `InfoElementSpec.`‘

An Information Model (`InfoModel`) is needed to allocate and initialize a new Template.

`add_spec`(*spec : InfoElementSpec*)

Adds a given InfoElementSpec *spec* to the Template.

Once the Template has been added to the session, it can not be altered.

`add_spec_list`(*list : List*)

Adds the given *list* of InfoElementSpec items to the `Template`.

Once the `Template` has been added to the `Session`, it can not be altered.

`add_element`(*name : str*)

Adds an Information Element with the given *name* to the Template. This function can be used as an alternative to add_spec.

This function creates an `InfoElementSpec` with the given element *name* and default length and adds it to the template.

`build_spec_list`()

Wallk through the `Template` to build the list of class:*InfoElementSpecs* contained in the `Template`.

This is typically used by a `Collector` or `Listener` for external templates that it has received from the `Exporter` or in a file.

`getIndexedIE`(*key: int*) → InfoElement

Returns the `InfoElement` at the given index *key* in the `Template`. Unlike the __getitem__ method which returns the `InfoElementSpec`, this method returns the `InfoElement` at a particular index.

`__contains__`([*name: str*, *ie: InfoElement*]) → bool

Determine if the given element is in the Template.

If *element* is a name, return True if an Information Element with the given name is included in the Template.

If *element* is an `InfoElement` or `InfoElementSpec`, return True if the element exists in the Template, False otherwise.

`__getitem__`(*key: str or int*)

Returns the `InfoElementSpec` with the given *name* or index

`__len__`() → int

Returns the number of elements in the template.

`scope`

Returns the scope associated with the template (integer). Scope can and should be changed if template is an Options Template.

`tid`

Returns the template ID associated with the template. Template ID can only be changed by adding a new template to a (:class: *Session*).

type

Returns *True* if template is an Information Element Type Information Template. Returns *False* otherwise. This element may not be changed.

Examples:

```
>>> tmpl = pyfixbuf.Template(model)
>>> spec = pyfixbuf.InfoElementSpec("sourceTransportPort")
>>> spec2 = pyfixbuf.InfoElementSpec("destinationTransportPort")
>>> tmpl.add_spec(spec)
>>> tmpl.add_spec(spec2)
>>> tmpl2 = pyfixbuf.Template(model)
>>> tmpl2.add_spec_list([pyfixbuf.InfoElementSpec("fooname"),
                    pyfixbuf.InfoElementSpec("barname")])
>>> tmpl2.scope = 2
>>> if "sourceTransportPort" in tmpl:
>>>    print "yes"
yes
```

1.5 Session

The state of an IPFIX Transport Session is maintained in the `Session` object. This includes all IPFIX Message Sequence Number tracking, and internal and external template management. Templates must be added before collecting or exporting any data.

class `pyfixbuf.`**`Session`**(*InfoModel*)

Creates an empty `Session` given an Information Model, *InfoModel*. A Session stores and manages all of the IPFIX Templates.

add_template(*template : Template*[, *template_id=0*]) → int

Adds the given *template* to the session with the optional *template_id*. This template will be added to both the internal and external templates. Use add_internal_template or add_external_template to be more selective on template usage.

If a *template_id* is not given or 0, libfixbuf will automatically choose one. *template_id* will be used for representing both the internal and external template.

Returns the *template_id* of the added template.

add_internal_template(*template : Template*[, *template_id=0*]) → int

Adds the given *template* as an internal template to the session with the optionally given *template_id*. An internal template determines how the data will be presented when transcoded.

If *template_id* is not set or 0, libfixbuf will automatically choose one.

Returns the *template_id* of the added template.

add_external_template(*template : Template*[, *template_id=0*]) → int

Adds the given *template* as an external template to the session with the optionally given *template_id*.

If *template_id* is not set or 0, libfixbuf will automatically choose one.

Returns the *template_id* of the added template.

decode_only(*list: List*)

This method is for IPFIX Collectors only. Only decode records in a list that have template IDs in the given *list*.

This does not apply for all incoming templates. This only applies for nested templates found in a SubTemplateMultiList or SubTemplateList.

ignore_templates (*list: List*)

This method is for IPFIX Collectors only. Ignore all templates with the template IDs in the given *list*.

This does not apply for all incoming templates. This only applies for nested templates found in a SubTemplateMultiList or SubTemplateList.

add_template_pair (*external_template_id : int*, *internal_template_id : int*)

This method is for IPFIX Collectors. This gives the collector control over how to transcode incoming templates in a SubTemplateMultiList (STML) or SubTemplateList (STL).

By default, libfixbuf transcodes each entry in the STML or STL with the external template it received, requiring the collector to free or clear any memory allocated for the list elements. The collector can change this behavior by adding a "template pair." For each entry in the STL or STML, if the entry has the given *external_template_id*, it will use the given *internal_template_id* to transcode the record. The *internal_template_id* must reference an internal template that was previously added to the session. If *internal_template_id* is 0, the entry will not be transcoded and will be ignored by libfixbuf.

Once a template pair has been added - the default is to ONLY decode entries that have an external_template_id in this template pair table. Therefore, any entries in a STML or STL that reference a template id not in this table will be dropped.

export_templates ()

Export the templates associated with this session. This is necessary for an exporting session and must be called before any records are appended to the `Buffer`. `Buffer` must already have a `Session` associated with it using `init_export`.

get_template (*template_id : int*[, *internal=False*]) → Template

Return the template with the given template_id. By default it returns the external template in the session with the given template_id. Returns None if the Template doesn't exist. The returned template cannot be modified. Set *internal* to True to retrieve the internal template with the given *template_id*.

Examples:

```
>>> session = pyfixbuf.Session(model)
>>> session.add_internal_template(289, tmpl)
>>> auto_id = session.add_external_template(0, tmpl)
>>> session.decode_only([256, 257])
```

1.6 Exporter

An Exporter maintains the information needed for its connection to a corresponding Collecting Process. An Exporter can be created to connect via the network using one of the supported IPFIX transport protocols, or to write to IPFIX files. Depending on the type of Exporter desired, one will use one of the following methods:

class `pyfixbuf.`**Exporter**

Creates an empty `Exporter`. Initialize the exporter using `init_file` or `init_net`.

init_file (*filename: str*)

Initializes the `Exporter` to write to the given *filename*.

init_net (*hostname: str*[, *transport="tcp"*, *port=4739*])

Initializes the `Exporter` to write to the given *hostname*, *port* over the given *transport*.

Given *hostname* may be a hostname or IP address.

Acceptable values for *transport* are "tcp" and "udp". Default is "tcp."

Given *port* must be greater than 1024. Default is 4739.

Examples:

```
>>> exporter = pyfixbuf.Exporter()
>>> exporter.init_file("/path/to/out.ipfix")
>>> exporter2 = pyfixbuf.Exporter()
>>> exporter2.init_net("localhost", "udp", 18000)
```

1.7 Collector

An `Collector` maintains the necessary information for the connection to a corresponding Exporting Process. A `Collector` is used for reading from an IPFIX file. See `Listener` for collecting IPFIX over a network.

class `pyfixbuf.`**`Collector`**

Creates an uninitialized `Collector`. An IPFIX Collector manages the file it is reading from. Initialize the collector using `init_file`.

`init_file`(*filename: str*)

Initialize the `Collector` to read from the given *filename*. *filename* should be the path to a valid IPFIX File.

Examples:

```
>>> collector = pyfixbuf.Collector()
>>> collector.init_file("path/to/in.ipfix")
```

1.8 Record

A `Record` is one of the "core" interaces to the IPFIX data through libfixbuf. This is the main object for manipulating the data prior to export and following import.

class `pyfixbuf.`**`Record`**(*model : InfoModel*[, *template=None*][, *record=None*])

Creates an empty Record given an `InfoModel`, *model*, and optionally a *template* and *record*.

The `Record` is returned from a collection `Buffer` or is added to an exporting `Buffer`.

When adding elements to a `Record`, the `Record` should match a `Template`. If the process is collecting, the `Record` should match the Internal Template. For an Exporting process, the `Record` should match the External Template, and there should be one `Record` for each External Template. A `Record` can not contain more Information Elements than it's associated *template*. Information Elements should be added to the `Record` in the same order as the `Template`.

If a *template* is given to the constructor,, all Information Elements that exist in the *template* will be added to the `Record` in the same order as they exist in the Template.

If a *record* is given, all Information Elements that exist in the *record* will be added to the `Record` in the same order as they exist in the *record*.

One element must exist in the `Record` before exporting any data.

A `Record` maintains internal dictionaries for the elements that it contains. For this reason, if a template contains more than 1 of the same Information Element, elements must be added using the `add_element` method in order to give alternate key names to elements that are the same.

A `Record` may also be accessed similar to a list.

add_element (*key_name : str*[, *type=0*, *element_name=None*, *length=0*])

Adds an Information Element with the given *key_name*, optional *type*, optional *element_name*, and optional reduced-length *length* to the `Record`.

If *key_name* is the same name as the defined `InfoElementSpec`, then a *type* is not necessary.

If the template contains more than one of the same Information Element, you must give an alternate *key_name*, *type*, and *element_name* to describe the other Information Elements.

A *type* 0 is a regular, fixed-length Information Element. Other valid types are VARLEN, BASICLIST, SUBTEMPLATELIST, and SUBTEMPLATEMULTILIST.

element_name is the defined Information Element name. This is only needed if the *key_name* is NOT a valid Information Element name and *type* = 0.

length is the reduced-length value for the Information Element. This can only be applied to certain data types and must be a smaller length than the default Information Element length. If set to 0, the length will default to the length provided by the InfoModel.

Elements must be added in the same order as they exist in the template.

Examples:

```
>>> my_rec = pyfixbuf.Record(model)
>>> my_rec.add_element("sourceTransportPort")
>>> my_rec.add_element("sourceTransportPort2", 0, "sourceTransportPort")
>>> my_rec.add_element("basicList")
>>> my_rec.add_element("basicList2", BASICLIST)
>>> my_rec.add_element("octetTotalCount", length=4)
```

In the above example, an empty `Record` was created. The corresponding template to the above `Record` would look something like:

```
>>> tmpl = Template(model)
>>> tmpl.add_spec_list([pyfixbuf.InfoElementSpec("sourceTransportPort"),
...                     pyfixbuf.InfoElementSpec("sourceTransportPort"),
...                     pyfixbuf.InfoElementSpec("basicList"),
...                     pyfixbuf.InfoElementSpec("basicList"),
...                     pyfixbuf.InfoElementSpec("octetTotalCount", 4)])
```

As you can see, we have two sourceTransportPort elements and two basicList elements. A basicList is a list of one or more of the same Information Element. The Information Element in the basicList does not have to be initialized until data is added to the `Record`.

Since we have two sourceTransportPort fields, we must give a *key_name* to one of the elements, in this case, sourceTransport2. Since sourceTransportPort2 is not a defined Information Element in the Information Model, the *element_name* must be given to the method.

Similarly, in order to access the dictionary of elements in the `Record`, we had to give the second basicList a *key_name*, basicList2. Since basicList2 is not a defined Information Element, it needs to be given the *type*, BASICLIST. Since *type* is not 0, it does not need an *element_name*.

add_element_list (*list : List*)

Adds the given *element_list*, a list of Information Element names to the `Record`. See above method `addElement`.

clear_all_lists ()

Clears all the lists in the top level of the `Record`.

Any nested lists must be accessed and cleared manually.

This is useful for a `Record` that contains mostly one level list items, such as YAF_HTTP_LIST.

clear()

Clears any memory allocated for the Record.

init_basic_list(*basic_list_key : str*[, *count=0*, *element_name=None*])

Initializes a basicList for export with the given *basic_list_key* name to a list of *count* elements. If a name is not given to the *element_name* keyword, it assumes the *basic_list_key* is a valid Information Element Name.

Examples:

```
>>> my_rec.add_element("bL", BASICLIST, "octetTotalCount")
>>> my_rec.add_element("basicList")
>>> my_rec.add_element("basicList2", BASICLIST)
>>> my_rec.init_basic_list("bL", 4)
>>> my_rec.init_basic_list("basicList", 3, "destinationTransportPort")
>>> my_rec.init_basic_list("basicList2", 2, "souceIPv4Address")
```

In the above example, we have initialized three basicLists. The first initializes a basicList of octetTotalCounts by adding the element as as basicList to the record. Later we initialize the basicList to 4 items. The second does the initialization of the type, destintationTransportPort, when calling init_basic_list as opposed to the first, which is done when the basicList is added to the record. The third, basicList2, is initialized to two sourceIPv4Adresses.

It is perfectly acceptable to initialize a list to 0 elements. All basicLists in the Record must be initialized before appending the Record to the Buffer.

A basicList may be initialized via this method, or by using the BL and setting the basicList element in the Record to the BL.

clear_basic_list(*basic_list_key : str*)

Clears the basicList. Frees any memory allocated for the list and should be called after the Record has been appended to the Buffer.

__getitem__(*key : str, int*)

Returns the value of the element with the given key. The return type depends on the Information Element type which was defined when initializing the InfoElement. *key* may be a string which corresponds to the key_name given to add_element() or the InfoElement name. *key* may also be an integer, which corresponds to the index in the Record.

Element Type	Return Type
UINT*, INT*	Long
FLOAT*	Float
MILLOSECONDS, MICROSECONDS	Long
NANOSECONDS, SECONDS	Long
OCTET_ARRAY	Byte Array
BASICLIST	BL
VARLEN	String
IP (v4 or v6)	IP String
MACADDR	MAC Address String xx:xx:xx:xx:xx:xx
SUBTEMPLATELIST	STL
SUBTEMPLATEMULTILIST	STML
Default (Undefined Type)	Byte Array

__setitem__(*key : str, value : int or str*)

Set the given element with name *key* to the given *value*. If the *value* is an IP Address, it will convert the String representation to an int. The *key* may be a string which represents either the InfoElement name or *key_name* given to add_element. The *key* may also be an integer, which is an index into the Record.

copy (*other : Record*)

Copies all the matching elements in this `Record` to the *other* `Record`.

is_list (*key_name : str*) → bool

Returns `True` or `False` depending on the type of the given *key_name*.

get_stl_list_entry (*key_name : str*) → STL

Gets the subTemplateList from the `Record` with the given *key_name* and returns a newly allocated `STL`.

A `STL` may also be accessed by using `__getitem__`.

get_stml_list_entry (*key_name: str*) → STML

Gets the subTemplateMultiList with the given *key_name* and returns a newly allocated `STML`.

A `STML` may also be retrieved by using __getitem__().

as_dict () → dictionary

Returns the `Record` as a dictionary.

__len__ () → int

Returns the number of elements in the `Record`.

__contains__ (*item: str*) → bool

Returns True if item is in the `Record`, False otherwise

set_template (*template: Template*)

If the `Record` was not initialized with a `Template`, this method is used to set the corresponding `Template` with the `Record`. A `Record` must have a `Template` associated with it when assigning a `Record` to a subTemplateList element.

Examples:

```
>>> tmpl = pyfixbuf.Template(model)
>>> tmpl.add_spec_list([pyfixbuf.InfoElementSpec("sourceTransportPort"),
...                     pyfixbuf.InfoElementSpec("destinationTransportPort")]
>>> my_rec = pyfixbuf.Record(model)
>>> my_rec.add_element("sourceTransportPort", "destinationTransportPort")
>>> my_rec["sourceTransportPort"] = 13
>>> my_rec["destinationTransportPort"] = 15
>>> my_rec.set_template(tmpl)
>>> other_rec["subTemplateList"] = [my_rec]
```

__iter__ () → Record

Iterate through the Record

next ()

Returns the next item in the `Record`

matches_template (*template: Template*) → bool

Returns True if the entries in the `Template` match the information element entries in the `Record`.

count (*key_name: str*) → int

Counts the occurrence of the *element_name* in the `Record`.

Examples:

```
>>> rec.add_element_list(["basicList", "basicList", "basicList"])
>>> rec.count("basicList")
3
>>> rec.count("sourceTransportPort")
0
```

1.9 Buffer

The `Buffer` implements a transcoding IPFIX Message buffer for both export and collection. The `Buffer` is one of the "core" interfaces to the fixbuf library. Each `Buffer` must be initialized to do either collecting or exporting.

class `pyfixbuf.`**`Buffer`**(*record : Record*)

Creates an uninitialized `Buffer` given a `Record`, *record*. A `Record` must be associated with the `Buffer` before retrieving or appending data to the Buffer. If *auto* is set, a `Template` will be auto-generated from the external template that is set on the `Buffer`. A `Record` will then be auto-generated to match the new `Template` that was set on the `Buffer`.

The `Buffer` must also be initialized for collection using `init_collection` or exporting `init_export` prior to calling next().

`init_collection`(*session : Session*, *collector : Collector*)

Initialize the `Buffer` for collection given the `Session`, *session*, and `Collector`, *collector*.

`init_export`(*session : Session*, *exporter : Exporter*)

Initialize the `Buffer` for Export given the `Session`, *session*, and `Exporter`, *exporter*.

`set_internal_template`(*v : int*)

The `Buffer` must have an internal template set on it before collecting or exporting. Set the internal template with the given template ID.

`set_export_template`(*v : int*)

The `Buffer` must have an export template set before appending any `Record` to the `Buffer`. This is how fixbuf will transcode the given `Record`. Set the external template with the given template ID.

`next_record`(*record : Record*) → Record

Get the next record on the buffer in the form of the given `Record`, *record*.

`next`() → Record

Returns the next `Record` in the buffer. Raises `StopIteration` Exception when done.

`__iter__`() → Buffer

Iterate through the buffer

`set_record`(*record : Record*)

Set the given *record* on the buffer

`next_template`() → Template

Retrieves the external template that will be used to read the next record from the buffer. If no next record is available, returns `None`.

`get_template`() → Template

Retrieves the external template that was used to read the last record from the buffer. If no record has been read, returns `None`.

`append`(*Record*[, *v : int*])

Appends the given *record* on the buffer. If a second argument, *length*, is given, append only the first *length* number of bytes to the buffer.

An internal and external template must be set on the buffer prior to appending an `Record`.

`write_ie_options_record`(*name: str*, *template: Template*)

Appends an Information Element Type Information Record on the Buffer. An Options Record will be written with information about the Information Element with the given *name*. *template* is the Information Element Type Options Template that was created by giving "type=1" to the Template constructor.

auto_insert()

Automatically insert any Information Elements that it receives Information Element Option Records for. It will only insert information elements that do not have private enterprise number of 0.

ignore_options(*ignore: bool*)

If *ignore* is set to True, the Buffer will ignore Options Templates and Records. By default, *ignore* is False, and the Buffer will return Options Records and the application must use next_template() to retrieve the `Template` set on the Buffer and determine if it is an Options Template.

emit()

Emit all records in the buffer.

Examples:

```
>>> buf = pyfixbuf.Buffer(my_rec)
>>> buf.init_collection(session, collector)
>>> buf.set_internal_template(999)
>>> for data in buf:
...     data = data.as_dict()
...     for key,value in data.items()
...         print key + ":" + str(value) + '\n'
```

Examples:

```
>>> buf = pyfixbuf.Buffer(my_rec)
>>> buf.init_export(session, exporter)
>>> buf.set_internal_template(999)
>>> buf.set_external_template(999)
>>> session.export_templates()
>>> while count < 10:
...       my_rec['sourceIPv4Address'] = "192.168.3.2"
...       my_rec['destinationIPv4Address'] = "192.168.4.5"
...       buf.append(my_rec)
>>> buf.emit()
```

Examples:

```
>> buf = pyfixbuf.Buffer(auto=True)
>> buf.init_collection(session, collector)
>> for data in buf:
...    data = data.as_dict()
...    for key,value in data.items()
...        print key + ":" + str(value) + '\n'
```

1.10 STML

A subTemplateMultiList is a list of zero or more instances of a structured data record, where the data records do not necessarily have to reference the same template. A subTemplateMultiList is made up of one or more `STMLEntry`. Each entry in the STML should (but are not required) have a different template associated with it. The data in the STML is accessed by iterating through each `STMLEntry` in the list and setting a `Record` on the `STMLEntry`.

class pyfixbuf.**STML**([*record=None*, *key_name=None*, *type_count=-1*])

A `STML` object represents a subTemplateMultiList.

If *record*, a `Record` object, and *key_name*, a string, are provided the `STML` object with *key_name* will be initialized in the given `Record`. It is only necessary to initialize and give a *type_count* if the subTemplateMultiList will be exported. All subTemplateMultiLists in an exported `Record` must be initialized. It is acceptable to initialize an STML to 0 list entries.

A STML must be initialized with *record* and *key_name* OR a *type_count*. This object can be used to set a subTemplateMultiList element in a Record.

The subTemplateMultiList is initialized to None unless it is given a *type_count*, in which case it will intialize the list and allocate memory in the given record.

type_count is the amount of different templates that the STML will contain. For example, if you plan to have an STML with entries of type Template ID 999 and 888, *type_count* would be 2. *type_count* would also be 2 even if both instances will use Template ID 999.

Examples:

```
>>> stml = my_rec["subTemplateMultiList"]   # sufficient for collection
>>> stml = pyfixbuf.STML(rec, "subTemplateMultiList", 3)  # STML with 3 entries for export
>>> stml = pyfixbuf.STML(type_count=2)
>>> stml = [record1, record2]
>>> stml2 = pyfixbuf.STML(type_count=3)
>>> stml2[0] = [record1, record2]
>>> stml2[1][0] = record3
>>> stml2[2].entry_init(record3, tmpl3, 0) #all entries must be init'd - even to 0.
>>> rec["subTemplateMultiList"] = stml
```

clear()
Clear the entries in the subTemplateMultiList and frees any memory allocated.

__iter__() → STML
Iterator for the SubTemplateMultiList

next() → STMLEntry
Returns the next SubTemplateMultiList Entry in the List

__len__() → int
Returns the number of entries in the subTemplateMultiList.

__contains__(*name: str*) → bool
Determine if item with element *name* is in the first entry in the STML.

__getitem__(*index: int*) → STMLEntry
Returns the STMLEntry at the *index*

Examples:

```
>>> entry = stml[0]
>>> stml[0].entry_init[record, template, 3]
```

__setitem__(*key: int*, *value: list*)
This sets an entry in the STML to the given list of Record objects. *value* must be a list. All Records in the list should have the same :class:Template.

Examples:

```
>>> stml[0] = [rec1, rec2, rec3, rec4]
```

semantic
The semantic value of the list of subTemplates.

Decode Examples:

```
>>> stml = my_rec["subTemplateMultiList"]
>>> for entry in stml:
...         if "tcpSequenceNumber" in entry:
...             entry.set_record(tcprec)
...             for tcp_record in entry:
```

```
...                 tcp_record = tcp_record.as_dict()
...                 for key,value in tcp_record.items()
...                     print key + ": " + str(value) + '\n'
```

Encode Examples:

```
>>> stml = STML(type_count=3)
>>> stml.entry_init(rec, template, 2) #init first entry to 2 with template
>>> rec["sourceTransportPort"] = 3
>>> rec["destinationTransportPort"] = 5
>>> stml[0][0] = rec
>>> rec["sourceTransportPort"] = 6
>>> rec["destinationTransportPort"] = 7
>>> stml[0][1] = rec
>>> stml[1][0] = rec2       #init second entry to 1 item using rec2
>>> stml[2].entry_init(rec3, template3, 0) #init third entry to 0
```

1.11 STMLEntry

Each `STML` consists of one or more `STMLEntry`. Each `STMLEntry` is associated with a template, and therefore should have a corresponding `Record`. An `STMLEntry` can contain zero or more instances of the associated `Record`.

class `pyfixbuf.`**`STMLEntry`**(*stml : STML*)

Creates an empty `STMLEntry` and associates it to the given `STML`, *stml*. There should be one `STMLEntry` for each different Template in the `STML`.

Each `STMLEntry` should be initialized using `entry_init` to associate a `Record` and `Template` with the entry.

`entry_init`(*record : Record*, *template : Template*[, *count=0*])

Initializes the `STMLEntry` to the given `Record`, *record*, *template*, and *count* instances of the *record* it will contain.

This should only be used for exporting a subTemplateMultiList. Entries in the `STML` must all be initialized, even if it is initialized to 0. This method is not necessary if a `Record` has a template associated with it. The application can simply set the `STMLEntry` to a list of `Records` and the `STMLEntry` will automatically be initialized.

Examples:

```
>>> stml = pyfixbuf.STML(my_rec, "subTemplateMultiList", 1)
>>> stml[0].entry_init(my_rec, template, 2)
>>> my_rec["sourceTransportPort"] = 3
>>> stml[0][0] = my_rec
>>> my_rec["sourceTransportPort"] = 5
>>> stml[0][1] = my_rec
```

`set_record`(*record : Record*)

Set the `Record`, *record*, on the `STMLEntry` to access its elements.

`__contains__`(*name : str*) → bool

Determine if the template associated with this STMLEntry contains the Information Element with the given *name*.

Alternatively, you can access the template ID associated with the STMLEntry to determine the type of `Record` that should be used to access the elements.

set_template (*template: Template*)

Assign a template to the STMLEntry. The given template must be a valid Template.

Use this method as an alternative to entry_init. This is only required if the Record that will be assigned to the STMLEntry was not created with a template. Using this method instead of entry_init will result in only allocating 1 item for the STMLEntry.

__iter__ () → STMLEntry

Iterator for the STML Entry

next () → Record

Retrieves the next Record in the STMLEntry.

If a Record has not been associated with this STMLEntry a Record will be auto generated using the current Template set on this STMLEntry.

__getitem__ (*item : str or int*) → Record

Get the Record at the given index. If item is a name of an information element, it will retrieve the value of that element for the first Record in the STMLEntry

__setitem__ (*key : int*, *value : Record*)

Set the entry item with the given key (index) to the given value. The value should a valid Record.

If the STMLEntry was not intialized, it will be initialized to 1 entry of the Record's Template.

__len__ () → int

The number of items in this entry.

template_id

The Template ID of the template that corresponds to this entry in the list.

Examples:

```
>>> stml = my_rec["subTemplateMultiList"]
>>> for entry in stml:
...     if "tcpSequenceNumber" in entry:
...        entry.set_record(tcp_rec)
...        for tcp_record in entry:
...            tcp_record = tcp_record.as_dict()
...            for key,value in tcp_record.items():
...                print key + ": " + str(value) + '\n'
...     elif entry.template_id == 0xCE00:
...        entry.set_record(dns_rec)
...
>>> stml.clear()
```

1.12 STL

A subTemplateList is a list of zero or more instances of a structured data type where each entry corresponds to a single template. Since a single template is associated with an STL, a Record must also be associated with the STL. Access each entry (a Record) in the list by iterating through the STL.

class pyfixbuf.**STL** ([*record=None*, *key_name=None*])

A STL represents a subTemplateList.

If *record*, a Record object, and *key_name*, a string, are provided, the subTemplateList for *key_name* in the given *record* are initialized, otherwise a generic STL will be initialized. Eventually a Template must be associated with the STL for encoding.

For decoding, a Record must be associated with the STL.

set_record (*record : Record*)
: Set the given *record* on the STL.

__contains__ (*name : str*) → bool
: Returns `True` if the element with the given *name* exists in the `Template` associated with the subTemplateList. Returns `False` if not present.

entry_init (*record : Record*, *template : Template*[, *count=0*])
: Initialize the STL to the given `Record`, *record*, and *template* to *count* entries.

  This method should only be used to export a `STL`.

  Each `STL` should be initialized before appending the `Record` to the `Buffer` even if it is initialized to 0.

  The record that contains the `STL` should not be modified after calling entry_init().

__iter__ () → STL
: Iterator for a `STL`

next () → Record
: This returns the next record in the `STL`

clear ()
: Clear all entries in the list. Nested elements should be accessed and freed before calling `clear`. Frees any memory previously allocated for the list.

__getitem__ (*item: int or str*) → Record
: Get the `Record` at the given index. If item is a name of an information element, it will retrieve the value of that element for the last `Record` accessed in the `STL`

__setitem__ (*key: int*, *value: Record*)
: Set the entry item with the given key (index) to the given value. The value should a valid `Record`. If the `STL` was not initialized via entry_init(), the `STL` will be initialized with the given `Record`‘s template and a count of 1.

__len__ ()
: The number of entries in the subTemplateList.

template_id
: The template ID used for this subTemplateList.

semantic
: The semantic value for the subTemplateList.

Decoding Examples:

```
>>> stl = rec["dnsList"]
>>> stl.set_record(dnsRecord)
>>> for dnsRecord in stl:
...     dnsRecord = dnsRecord.as_dict()
...     for key,value in dnsRecord.items():
...         print key + ": " + str(value) + ’\n’
... stl.clear()
```

Encoding Examples:

```
>>> stl = STL()
>>> stl.entry_init(dnsRecord, dnsTemplate, 2)
>>> dnsRecord["dnsQName"] = "google.com"
>>> dnsRecord["rrType"] = 1
>>> stl[0] = dnsRecord
>>> dnsRecord["dnsQName"] = "ns.google.com"
>>> dnsRecord["rrType"] = 2
>>> stl[1] = dnsRecord
>>> rec["subTemplateList"] = stl
```

1.13 BL

A basicList is a list of zero or more instances of an Information Element. Examples include a list of port numbers, or a list of host names. The BL object acts similar to a Python list with additional attributes.

class `pyfixbuf.`**`BL`**(*model : InfoModel*, *element : str*, *InfoElementSpec*[, *count=0*, *semantic=0*])

A `BL` represents a basicList.

A basicList is a list of zero or more instances of an Information Element.

A basicList can be initialized through a `Record` via init_basic_list(), or by creating a `BL` object.

The constructor requires an `InfoModel` *model*, and a `InfoElementSpec`, `InfoElement`, or string *element*. Additionally, it takes an optional integer *count* which represents the number of elements in the list, and an optional integer *semantic* to express the relationship among the list items.

All basicLists in a `Record` must be initialized (even to 0) before appending a `Record` to a `Buffer`.

Examples:

```
>>> rec.add_element("basicList", BASICLIST)
>>> rec.add_element("basicList2", BASICLIST)
>>> bl = BL(model, "sourceTransportPort", 2)
>>> bl[0] = 80
>>> bl[1] = 23
>>> rec["basicList"] = bl
>>> rec.init_basic_list("basicList2", 4, "octetTotalCount")
>>> rec["basicList2"] = [99, 101, 104, 23]
```

`__len__`()

The number of entries in the basicList.

`__iter__`()

Iterate through the basicList

`next`()

Returns the next item in the basicList

`__getitem__`(*index: int*)

Returns the value for the *index* in the basicList

`__setitem__`(*key: int*, *value: str*, *int*)

`copy`(*other : list*)

Copy all the items in the list to the `BL`. This will only copy up to the length of the `BL`.

`__contains__`(*item : str*, *int*) → bool

Returns True if item is in the `BL`, False otherwise

`__str__`() → str

`__eq__`(*other : list*) → bool

`clear`()

Clears and frees the basicList data.

`semantic`

The semantic value for the basicList.

element
The element associated with the basicList. This returns an InfoElement.

Decoding Examples:

```
>>> bl = rec["basicList"]
>>> for items in bl:
...    print str(items) + '\n'
... bl.clear()
```

Encoding Examples:

```
>>> bl = BL(model, "httpUserAgent", 2)
>>> bl[0] = "Mozilla/Firefox"
>>> bl[1] = "Safari5.0"
>>> rec["basicList"] = bl
>>> if "Safari5.0" in bl:
...     print "Apple"
Apple
>>> print bl
["Mozilla/Firefox", "Safari5.0"]
```

1.14 Listener

The Listener manages the passive collection used to listen for connections from Exporting Processes.

class `pyfixbuf.`**`Listener`**(*session : Session*, *hostname : str*[, *transport="tcp"*, *port=4739*])
Create a `Listener` given a *session*, transport protocol, *transport*, *hostname*, and *port* to listen on.

session must be a valid instance of `Session`.

hostname may be a hostname or IP Address.

transport may contain "tcp" or "udp". Default is "tcp."

port should be greater than 1024. Default is 4739.

Examples:

```
>>> listener = Listener(session, hostname="localhost", port=18000)
```

`wait`([*record : Record*])
Wait for a connection on the set host and port. Returns a newly allocated `Buffer`.

If a `Record` is given to `wait` then the returned `Buffer` will already be associated with an `Record`.

If no `Record` is given, you must use `set_record` on the `Buffer` before accessing the elements.

After receiving the `Buffer` you must set the internal template on the returned `Buffer` using `set_internal_template` before accessing the data.

Examples:

```
>>> buf = listener.wait()
>>> buf.set_record(my_rec)
>>> buf.set_internal_template(999)
>>> for data in buf:
>>> ...
```

1.15 Pre-defined Information Element Lists

pyfixbuf groups the YAF-defined Information Elements, CERT PEN 6871, by protocol. YAF_LIST and YAF_STATS are necessary for collecting default input streams from YAF. Adding the following lists to the InfoModel will result in adding the following Information Elements to the InfoModel.

1.15.1 YAF_LIST

Information Element	ID	TYPE	Description
initialTCPFlags	14	UINT8	Initial sequence number of the forward direction of the flow
unionTCPFlags	15	UINT8	Union of TCP flags of all packets other than the initial packet in the forward direction of the flow
reverseFlowDeltaMilliseconds	21	UINT32	Difference in time in milliseconds between first packet in forward direction and first packet in reverse direction
silkAppLabel	33	UINT16	Application label, defined as the primary well-known port associated with a given application.
osName	36	STRING	p0f OS Name for the forward flow based on the SYN packet and p0f SYN Fingerprints.
payload	36	OCTET ARRAY	Initial n bytes of forward direction of flow payload.
osVersion	37	STRING	p0f OS Version for the forward flow based on the SYN packet and p0f SYN Fingerprints.
firstPacketBanner	38	OCTET ARRAY	IP and transport headers for first packet in forward direction to be used for external OS Fingerprinters.
secondPacketBanner	39	OCTET ARRAY	IP and transport headers for first packet in forward direction to be used for external OS Fingerprinters.
flowAttributes	40	UINT16	Miscellaneous flow attributes for the forward direction of the flow
osFingerPrint	107	STRING	p0f OS Fingerprint for the forward flow based on the SYN packet and p0f SYN fingerprints.
yafFlowKeyHash	106	UINT32	The 32 bit hash of the 5-tuple and VLAN that is used as they key to YAF's internal flow table.

1.15.2 YAF_STATS_LIST

Information Element	ID	TYPE	Description
expiredFragmentCount	100	UINT32	Total amount of fragments that have been expired since yaf start time.
assembledFragmentCount	101	UINT32	Total number of packets that been assembled from a series of fragments since yaf start time.
meanFlowRate	102	UINT32	The mean flow rate of the yaf flow sensor since yaf start time, rounded to the nearest integer.
meanPacketRate	103	UINT32	The mean packet rate of the yaf flow sensor since yaf start time, rounded to the nearest integer.
flowTableFlushEventCount	104	UINT32	Total number of times the yaf flow table has been flushed since yaf start time.
flowTablePeakCount	105	UINT32	The maximum number of flows in the yaf flow table at any one time since yaf start time.

1.15.3 YAF_FLOW_STATS_LIST

Information Element	ID	TYPE	Description
smallPacketCount	500	UINT32	The number of packets that contain less than 60 bytes of payload.
nonEmptyPacketCount	501	UINT32	The number of packets that contain at least 1 byte of payload.
dataByteCount	502	UINT64	Total bytes transferred as payload.
averageInterarrivalTime	503	UINT64	Average number of milliseconds between packets.
standardDeviationInterarrivalTime	504	UINT64	Standard deviation of the interarrival time for up to the first ten packets.
firstNonEmptyPacketSize	505	UINT16	Payload length of the first non-empty packet.
maxPacketSize	506	UINT16	The largest payload length transferred in the flow.
firstEightNonEmptyPacketDirections	507	UINT8	Represents directionality for the first 8 non-empty packets. 0 for forward direction, 1 for reverse direction.
standardDeviationPayloadLength	508	UINT16	The standard deviation of the payload length for up to the first 10 non empty packets.
tcpUrgCount	509	UINT32	The number of TCP packets that have the URGENT Flag set.
largePacketCount	510	UINT32	The number of packets that contain at least 220 bytes of payload.

1.15.4 YAF_HTTP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
httpServerString	110	STRING
httpUserAgent	111	STRING
httpGet	112	STRING
httpConnection	113	STRING
httpVersion	114	STRING
httpReferer	115	STRING
httpLocation	116	STRING
httpHost	117	STRING
httpContentLength	118	STRING
httpAge	119	STRING
httpAccept	120	STRING
httpAcceptLanguage	121	STRING
httpContentType	122	STRING
httpResponse	123	STRING
httpCookie	220	STRING
httpSetCookie	221	STRING
httpAuthorization	252	STRING
httpVia	253	STRING
httpX-Forwarded-For	254	STRING
httpRefresh	256	STRING
httpIMEI	257	STRING
httpIMSI	258	STRING
httpMSISDN	259	STRING
httpSubscriber	260	STRING
httpExpires	255	STRING
httpAcceptCharset	261	STRING

Continued on next page

Table 1.1 – continued from previous page

Information Element	ID	TYPE
httpAcceptEncoding	262	STRING
httpAllow	263	STRING
httpDate	264	STRING
httpExpect	265	STRING
httpFrom	266	STRING
httpProxyAuthentication	267	STRING
httpUpgrade	268	STRING
httpWarning	269	STRING
httpDNT	270	STRING
httpX-Forwarded-Proto	271	STRING
httpX-Forwarded-Host	272	STRING
httpX-Forwarded-Server	273	STRING
httpX-DeviceID	274	STRING
httpX-Profile	275	STRING
httpLastModified	276	STRING
httpContentEncoding	277	STRING
httpContentLanguage	278	STRING
httpContentLocation	279	STRING
httpX-UA-Compatible	280	STRING

1.15.5 YAF_SLP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
slpVersion	128	UINTE8
slpMessageType	129	UINT8
slpString	130	STRING

1.15.6 YAF_FTP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
ftpReturn	131	STRING
ftpUser	132	STRING
ftpPass	133	STRING
ftpType	134	STRING
ftpRespCode	135	STRING

1.15.7 YAF_IMAP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
imapCapability	136	STRING
imapLogin	137	STRING
imapStartTLS	138	STRING
imapAuthenticate	139	STRING
imapCommand	140	STRING
imapExists	141	STRING
imapRecent	142	STRING

1.15.8 YAF_RTSP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
rtspURL	143	STRING
rtspVersion	144	STRING
rtspReturnCode	145	STRING
rtspContentLength	146	STRING
rtspCommand	147	STRING
rtspContentType	148	STRING
rtspTransport	149	STRING
rtspCSeq	150	STRING
rtspLocation	151	STRING
rtspPacketsReceived	152	STRING
rtspUserAgent	153	STRING
rtspJitter	154	STRING

1.15.9 YAF_SIP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
sipInvite	155	STRING
sipCommand	156	STRING
sipVia	157	STRING
sipMaxForwards	158	STRING
sipAddress	159	STRING
sipContentLength	160	STRING
sipUserAgent	161	STRING

1.15.10 YAF_SMTP_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
smtpHello	162	STRING
smtpFrom	163	STRING
smtpTo	164	STRING
smtpContentType	165	STRING
smtpSubject	166	STRING
smtpFilename	167	STRING
smtpContentDisposition	168	STRING
smtpResponse	169	STRING
smtpEnhanced	170	STRING
smtpSize	222	STRING
smtpDate	251	STRING

1.15.11 YAF_DNS_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
dnsQueryResponse	174	UINT8
dnsQRType	175	UINT16
dnsAuthoritative	176	UINT8
dnsNXDomain	177	UINT8
dnsRRSection	178	UINT8
dnsQName	179	STRING
dnsCName	180	STRING
dnsMXPreference	181	UINT16
dnsMXExchange	182	STRING
dnsNSDName	183	STRING
dnsPTRDName	184	STRING
dnsTTL	199	UINT32
dnsTXTData	208	STRING
dnsSOASerial	209	UINT32
dnsSOARefresh	210	UINT32
dnsSOARetry	211	UINT32
dnsSOAExpire	212	UINT32
dnsSOAMinimum	213	UINT32
dnsSOAMName	214	STRING
dnsSOARName	215	STRING
dnsSRVPriority	216	UINT16
dnsSRVWeight	217	UINT16
dnsSRVPort	218	UINT16
dnsSRVTarget	219	STRING
dnsID	226	UINT16
dnsAlgorithm	227	UINT8
dnsKeyTag	228	UINT16
dnsSigner	229	STRING
dnsSignature	230	OCTET ARRAY
dnsDigest	231	OCTET ARRAY
dnsPublicKey	232	OCTET ARRAY
dnsSalt	233	OCTET ARRAY
dnsHashData	234	OCTET ARRAY

Continued on next page

Table 1.2 – continued from previous page

Information Element	ID	TYPE
dnsIterations	235	UINT16
dnsSignatureExpiration	236	UINT32
dnsSignatureInception	237	UINT32
dnsDigestType	238	UINT8
dnsLabels	239	UINT8
dnsTypeCovered	240	UINT16
dnsFlags	241	UINT16

1.15.12 YAF_SSL_LIST

Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
sslCipher	185	UINT32
sslClientVersion	186	UINT8
sslServerCipher	187	UINT32
sslCompressionMethod	188	UINT8
sslCertVersion	189	UINT8
sslCertSignature	190	STRING
sslCertIssuerCountryName	191	STRING
sslCertIssuerOrgName	192	STRING
sslCertIssuerOrgUnitName	193	STRING
sslCertIssuerZipCode	194	STRING
sslCertIssuerState	195	STRING
sslCertIssuerCommonName	196	STRING
sslCertIssuerLocalityName	197	STRING
sslCertIssuerStreetAddress	198	STRING
sslCertSubCountryName	200	STRING
sslCertSubOrgName	201	STRING
sslCertSubOrgUnitName	202	STRING
sslCertSubZipCode	203	STRING
sslCertSubState	204	STRING
sslCertSubCommonName	205	STRING
sslCertSubLocalityName	206	STRING
sslCertSubStreetAddress	207	STRING
sslCertSerialNumber	208	STRING
sslObjectType	245	UINT8
sslObjectValue	246	STRING
sslCertValidityNotBefore	247	STRING
sslCertValidityNotAfter	248	STRING
sslCertPublicKeyAlgorithm	249	STRING
sslCertPublicKeyLength	250	UINT16
sslRecordVersion	288	UINT16

1.15.13 YAF_DPI_LIST

This list contains miscellaneous Information Elements from the remaining protocols YAF decodes. Descriptions of each Information Element can be found at http://tools.netsa.cert.org/yaf/yafdpi.html.

Information Element	ID	TYPE
mysqlUsername	223	STRING
mysqlCommandCode	224	UINT8
mysqlCommandText	225	STRING
pop3TextMessage	124	STRING
ircTextMessage	125	STRING
tftpFilename	126	STRING
tftpMode	127	STRING
dhcpFingerPrint	242	STRING
dhcpVendorCode	243	STRING
dnp3SourceAddress	281	UINT16
dnp3DestinationAddress	282	UINT16
dnp3Function	283	UINT8
dnp3ObjectData	284	OCTET_ARRAY
modbusData	285	OCTET_ARRAY
ethernetIPData	286	OCTET_ARRAY
rtpPayloadType	287	UINT8

CHAPTER

TWO

PYFIXBUF EXAMPLES

2.1 Collector Example

The pyfixbuf API aims to follow the original C library. The following example follows the traditional method of collecting IPFIX with libfixbuf:

1. Create an information model
2. Add Private Enterprise Number (PEN) Information Elements to the model.
3. Create an IPFIX template(s).
4. Define what the template(s) will contain.
5. Add the elements to the template.
6. Create an IPFIX collector (file vs TCP vs UDP)
7. Create a session.
8. Add the template(s) to the session.
9. Create an incoming data buffer.
10. Associate the collector and the session to the buffer.
11. Set the internal template on the buffer.
12. Data is read into Records.

```
#!/usr/bin/env python

import sys
# Import pyfixbuf
import pyfixbuf

# Import times from netsa-python for nice timestamp formats
from netsa.data.times import *

# Test that the number of arguments is correct

if ( len (sys.argv) != 2):
   print "Must supply ONLY an IPFIX file to read"
   sys.exit()

# Create an InfoModel
infomodel = pyfixbuf.InfoModel()
```

```
# Add YAF basic and stats information elements
infomodel.add_element_list(pyfixbuf.YAF_LIST)
infomodel.add_element_list(pyfixbuf.YAF_STATS_LIST)

# Create a Template
tmpl = pyfixbuf.Template(infomodel)

# Create a Stats Template to receive YAF Stats (Options) Records
stats_tmpl = pyfixbuf.Template(infomodel)

# Add some elements to the internal template
# This is a normal YAF flow record

data_list = [pyfixbuf.InfoElementSpec("flowStartMilliseconds"),
          pyfixbuf.InfoElementSpec("flowEndMilliseconds"),
          pyfixbuf.InfoElementSpec("octetTotalCount"),
          pyfixbuf.InfoElementSpec("reverseOctetTotalCount"),
          pyfixbuf.InfoElementSpec("packetTotalCount"),
          pyfixbuf.InfoElementSpec("reversePacketTotalCount"),
          pyfixbuf.InfoElementSpec("sourceIPv4Address"),
          pyfixbuf.InfoElementSpec("destinationIPv4Address"),
          pyfixbuf.InfoElementSpec("sourceTransportPort"),
          pyfixbuf.InfoElementSpec("destinationTransportPort"),
          pyfixbuf.InfoElementSpec("flowAttributes"),
          pyfixbuf.InfoElementSpec("reverseFlowAttributes"),
          pyfixbuf.InfoElementSpec("protocolIdentifier"),
          pyfixbuf.InfoElementSpec("flowEndReason"),
          pyfixbuf.InfoElementSpec("silkAppLabel"),
          pyfixbuf.InfoElementSpec("subTemplateMultiList")]

tmpl.add_spec_list(data_list)

# Add elements to the stats template (this is a subset of the YAF stats)

stats_list = [pyfixbuf.InfoElementSpec("exportedFlowRecordTotalCount"),
              pyfixbuf.InfoElementSpec("packetTotalCount"),
              pyfixbuf.InfoElementSpec("droppedPacketTotalCount"),
              pyfixbuf.InfoElementSpec("ignoredPacketTotalCount")]

stats_tmpl.add_spec_list(stats_list)

# Create a collector

collector = pyfixbuf.Collector()

# Initialize the collector to read an IPFIX file

collector.init_file(sys.argv[1])

# create a session

session = pyfixbuf.Session(infomodel)

# Add your data template to the session

session.add_internal_template(tmpl, 999)

# Add the stats template to the session
```

```
session.add_internal_template(stats_tmpl, 911)

# Create a Record for each Template and/or each SubTemplate
# The following rec will contain all the elements in the data template
rec = pyfixbuf.Record(infomodel, tmpl)

# The following rec will contain all the elements in the stats template
statsrec = pyfixbuf.Record(infomodel, stats_tmpl)

# Create a TCP Record, since YAF exports TCP information in the
# subTemplateMultiList by default

tcprec = pyfixbuf.Record(infomodel)

# Since we don't need a template for this TCP Record because
# it belongs in the subTemplateMultiList, we have to add
# the TCP elements using the addElement method

tcp_elements = ["tcpSequenceNumber", "initialTCPFlags", "unionTCPFlags",
          "reverseInitialTCPFlags", "reverseUnionTCPFlags", "reverseTcpSequenceNumber"]

tcprec.add_element_list(tcp_elements)

# create a new buffer for collection - rec matches our internal template
buf = pyfixbuf.Buffer(rec)

# initialize the buffer for collection
buf.init_collection(session, collector)

# set the internal template on the buffer
buf.set_internal_template(999)

# Now we can get the elements from the buffer

for data in buf:
    data = data.as_dict()
    print "------FLOW-------"
    for key,value in data.items():
    if (key == "flowStartMilliseconds" or key == "flowEndMilliseconds"):
       # use netsa-python to print times
       print key + ": " + str(make_datetime(value/1000))
    # print every element that is not a subtemplatemultilist
    elif key != "subTemplateMultiList":
       print key + ": " + str(value)

    # retrieve STML
    stml = data["subTemplateMultiList"]
    # Iterate through entries in STML
    for entry in stml:
        # Is it a TCP Template?
        if "tcpSequenceNumber" in entry:
           # set the tcprec on the entry
           entry.set_record(tcprec)
           # iterate through records in this entry of the stml
           for record in entry:
               record = record.as_dict()
               for key,value in record.items():
                   print key + ": " + str(value)
    # clear the STML
    stml.clear()

    # Now check to see if the next record is a stats record
    # by checking the next template on the buffer

    tmpl_next = buf.next_template()
    # if a template has scope - it's an options template
    if ( tmpl_next.scope ):
       # Set the internal template to the stats template
       buf.set_internal_template(911)
       # get the next record in the buffer as a stats record
       stats = buf.next_record(statsrec)
       print "----STATS----"
       if (stats != None):
          stats = stats.as_dict()
          # print all the items in stats
          for key,value in stats.items():
              print key + ": " + str(value)
         # Set the internal template back to the data template
       buf.set_internal_template(999)
```

It may be the case that the IPFIX data can change often and the application needs to be able to collect everything that the records contain. In that case, pyfixbuf can be used to build Records on the fly based on the templates that it receives. This is slightly different than the traditional way of reading IPFIX. Typically, the application knows what kind of data it wants and libfixbuf will populate only the fields the application cares about. In the following example, the application wants to view the contents of every IPFIX record in the file.

```
#!/usr/bin/env python

import sys
# Import pyfixbuf
import pyfixbuf

# Import times from netsa-python for nice timestamp formats
from netsa.data.times import *

# Test that the number of arguments is correct

if ( len (sys.argv) != 2):
   print "Must supply ONLY an IPFIX file to read"
   sys.exit()

# Create an InfoModel
infomodel = pyfixbuf.InfoModel()

# Create a collector

collector = pyfixbuf.Collector()

# Initialize the collector to read an IPFIX file

collector.init_file(sys.argv[1])

# create a session

session = pyfixbuf.Session(infomodel)
```

```
 # create a new buffer for collection
 buf = pyfixbuf.Buffer(auto=True)

# initialize the buffer for collection
 buf.init_collection(session, collector)

 for data in buf:

      print "------FLOW %d-------" % count
      for key,value in data.as_dict().items():
          if (key == "flowStartMilliseconds" or key == "flowEndMilliseconds"):
             # use netsa-python to print times
             print key + ": " + str(make_datetime(value/1000))
          # print every element that is not a subtemplatemultilist
          elif key != "subTemplateMultiList":
               print str(key) + ": " + str(value)
      # retrieve STML
      if "subTemplateMultiList" in data:
         stml = data["subTemplateMultiList"]
         # Iterate through entries in STML
         for entry in stml:
             for record in entry:
                 record = record.as_dict()
                 for key,value in record.items():
                     if key != "subTemplateList":
                        print str(key) + ": " + str(value)
                 if "subTemplateList" in record:
                     stl = record["subTemplateList"]
                     for sub in stl:
                         for key, value in sub.as_dict().items():
                             print str(key) + ": " + str(value)
                     stl.clear()

          # clear the STML
          stml.clear()
      count += 1
```

See the other examples included with the pyfixbuf package in “samples.”

INDEX

Symbols

__contains__() (pyfixbuf.BL method), 19
__contains__() (pyfixbuf.Record method), 12
__contains__() (pyfixbuf.STL method), 18
__contains__() (pyfixbuf.STML method), 15
__contains__() (pyfixbuf.STMLEntry method), 16
__contains__() (pyfixbuf.Template method), 6
__eq__() (pyfixbuf.BL method), 19
__getitem__() (pyfixbuf.BL method), 19
__getitem__() (pyfixbuf.Record method), 11
__getitem__() (pyfixbuf.STL method), 18
__getitem__() (pyfixbuf.STML method), 15
__getitem__() (pyfixbuf.STMLEntry method), 17
__getitem__() (pyfixbuf.Template method), 6
__iter__() (pyfixbuf.BL method), 19
__iter__() (pyfixbuf.Buffer method), 13
__iter__() (pyfixbuf.Record method), 12
__iter__() (pyfixbuf.STL method), 18
__iter__() (pyfixbuf.STML method), 15
__iter__() (pyfixbuf.STMLEntry method), 17
__len__() (pyfixbuf.BL method), 19
__len__() (pyfixbuf.Record method), 12
__len__() (pyfixbuf.STL method), 18
__len__() (pyfixbuf.STML method), 15
__len__() (pyfixbuf.STMLEntry method), 17
__len__() (pyfixbuf.Template method), 6
__setitem__() (pyfixbuf.BL method), 19
__setitem__() (pyfixbuf.Record method), 11
__setitem__() (pyfixbuf.STL method), 18
__setitem__() (pyfixbuf.STML method), 15
__setitem__() (pyfixbuf.STMLEntry method), 17
__str__() (pyfixbuf.BL method), 19

A

add_element() (pyfixbuf.InfoModel method), 5
add_element() (pyfixbuf.Record method), 9
add_element() (pyfixbuf.Template method), 6
add_element_list() (pyfixbuf.InfoModel method), 5
add_element_list() (pyfixbuf.Record method), 10
add_external_template() (pyfixbuf.Session method), 7
add_internal_template() (pyfixbuf.Session method), 7
add_options_element() (pyfixbuf.InfoModel method), 5
add_spec() (pyfixbuf.Template method), 6
add_spec_list() (pyfixbuf.Template method), 6
add_template() (pyfixbuf.Session method), 7
add_template_pair() (pyfixbuf.Session method), 8
append() (pyfixbuf.Buffer method), 13
as_dict() (pyfixbuf.Record method), 12
auto_insert() (pyfixbuf.Buffer method), 13

B

BL (class in pyfixbuf), 19
Buffer (class in pyfixbuf), 13
build_spec_list() (pyfixbuf.Template method), 6

C

clear() (pyfixbuf.BL method), 19
clear() (pyfixbuf.Record method), 10
clear() (pyfixbuf.STL method), 18
clear() (pyfixbuf.STML method), 15
clear_all_lists() (pyfixbuf.Record method), 10
clear_basic_list() (pyfixbuf.Record method), 11
Collector (class in pyfixbuf), 9
copy() (pyfixbuf.BL method), 19
copy() (pyfixbuf.Record method), 11
count() (pyfixbuf.Record method), 12

D

decode_only() (pyfixbuf.Session method), 7
description (pyfixbuf.InfoElement attribute), 4

E

element (pyfixbuf.BL attribute), 19
emit() (pyfixbuf.Buffer method), 14
endian (pyfixbuf.InfoElement attribute), 4
ent (pyfixbuf.InfoElement attribute), 3
entry_init() (pyfixbuf.STL method), 18
entry_init() (pyfixbuf.STMLEntry method), 16
export_templates() (pyfixbuf.Session method), 8
Exporter (class in pyfixbuf), 8

G

get_element() (pyfixbuf.InfoModel method), 5

get_element_length() (pyfixbuf.InfoModel method), 5
get_element_type() (pyfixbuf.InfoModel method), 5
get_stl_list_entry() (pyfixbuf.Record method), 12
get_stml_list_entry() (pyfixbuf.Record method), 12
get_template() (pyfixbuf.Buffer method), 13
get_template() (pyfixbuf.Session method), 8
getIndexedIE() (pyfixbuf.Template method), 6

I

id (pyfixbuf.InfoElement attribute), 3
ignore_options() (pyfixbuf.Buffer method), 14
ignore_templates() (pyfixbuf.Session method), 7
InfoElement (class in pyfixbuf), 2
InfoElementSpec (class in pyfixbuf), 4
InfoModel (class in pyfixbuf), 4
init_basic_list() (pyfixbuf.Record method), 11
init_collection() (pyfixbuf.Buffer method), 13
init_export() (pyfixbuf.Buffer method), 13
init_file() (pyfixbuf.Collector method), 9
init_file() (pyfixbuf.Exporter method), 8
init_net() (pyfixbuf.Exporter method), 8
is_list() (pyfixbuf.Record method), 12

L

length (pyfixbuf.InfoElement attribute), 3
length (pyfixbuf.InfoElementSpec attribute), 4
Listener (class in pyfixbuf), 20

M

matches_template() (pyfixbuf.Record method), 12
max (pyfixbuf.InfoElement attribute), 3
min (pyfixbuf.InfoElement attribute), 3

N

name (pyfixbuf.InfoElement attribute), 3
name (pyfixbuf.InfoElementSpec attribute), 4
next() (pyfixbuf.BL method), 19
next() (pyfixbuf.Buffer method), 13
next() (pyfixbuf.Record method), 12
next() (pyfixbuf.STL method), 18
next() (pyfixbuf.STML method), 15
next() (pyfixbuf.STMLEntry method), 17
next_record() (pyfixbuf.Buffer method), 13
next_template() (pyfixbuf.Buffer method), 13

P

pyfixbuf (module), 1

R

Record (class in pyfixbuf), 9
reversible (pyfixbuf.InfoElement attribute), 4
RFC
    RFC 5610, 2, 6

S

scope (pyfixbuf.Template attribute), 6
semantic (pyfixbuf.BL attribute), 19
semantic (pyfixbuf.InfoElement attribute), 3
semantic (pyfixbuf.STL attribute), 18
semantic (pyfixbuf.STML attribute), 15
Session (class in pyfixbuf), 7
set_export_template() (pyfixbuf.Buffer method), 13
set_internal_template() (pyfixbuf.Buffer method), 13
set_record() (pyfixbuf.Buffer method), 13
set_record() (pyfixbuf.STL method), 17
set_record() (pyfixbuf.STMLEntry method), 16
set_template() (pyfixbuf.Record method), 12
set_template() (pyfixbuf.STMLEntry method), 16
STL (class in pyfixbuf), 17
STML (class in pyfixbuf), 14
STMLEntry (class in pyfixbuf), 16

T

Template (class in pyfixbuf), 5
template_id (pyfixbuf.STL attribute), 18
template_id (pyfixbuf.STMLEntry attribute), 17
tid (pyfixbuf.Template attribute), 6
type (pyfixbuf.InfoElement attribute), 3
type (pyfixbuf.Template attribute), 6

U

units (pyfixbuf.InfoElement attribute), 3

W

wait() (pyfixbuf.Listener method), 20
write_ie_options_record() (pyfixbuf.Buffer method), 13