ZLogging

Release 0.1.0rc1

Jarry Shaw

Feb 14, 2020

CONTENTS

CHAPTER

ONE

BRO/ZEEK LOGGING FRAMEWORK FOR PYTHON

1.1 Table of Contents

1.1.1 Dumpers

Predefined Dumpers

Bro/Zeek log dumper.

class zlogging.dumper.**JSONWriter**
Bases: *zlogging.dumper.BaseWriter*

JSON log writer.

property format
str: Log file format.

write_file(*file*, *data*)
Write log file.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The file offset after writing.

Return type int

write_line(*file*, *data*, *lineno=0*)
Write log line as one-line record.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The file offset after writing.

Raises ***JSONWriterError*** – If failed to serialise data as JSON.

Return type int

dump_file(*data*)
Serialise records to a log line.

Parameters **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The converted log string.

Return type str

dump_line(*data*, *lineno=0*)
Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The converted log string.

Raises ***JSONWriterError*** – If failed to serialise data as JSON.

Return type str

class zlogging.dumper.**ASCIIWriter**(*separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*)
Bases: *zlogging.dumper.BaseWriter*

ASCII log writer.

Parameters

- **separator** (str or bytes, optional) – Field separator when writing log lines.
- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.

Variables

- **separator** (*bytes*) – Field separator when writing log lines.
- **str_separator** (*str*) – Field separator when writing log lines.
- **empty_field** (*bytes*) – Placeholder for empty field.
- **str_empty_field** (*str*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **str_unset_field** (*str*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for set/list fields.
- **str_set_separator** (*str*) – Separator for set/list fields.

property format
str: Log file format.

write_file(*file*, *data*)
Write log file.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The file offset after writing.

Return type int

write_line(*file*, *data*, *lineno=0*)
Write log line as one-line record.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The file offset after writing.

Raises ***ASCIIWriterError*** – If failed to serialise `data` as ASCII.

Return type int

write_head(*file*, *data=None*)
Write header fields of ASCII log file.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (*Model*, optional) – Log record.

Returns The file offset after writing.

Return type int

write_tail(*file*)
Write trailing fields of ASCII log file.

Parameters **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.

Returns The file offset after writing.

Return type int

dump_file(*data*, *name=None*)
Serialise records to a log line.

Parameters

- **data** (`Iterable` of *Model*) – Log records as an `Iterable` of *Model* per line.
- **name** (*Optional[str]*) – Log file name.

Returns The converted log string.

Return type str

dump_line(*data*, *lineno=0*)
Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The converted log string.

Raises ***ASCIIWriterError*** – If failed to serialise `data` as ASCII.

Return type str

dump_head(*data=None*, *name=None*)
Serialise header fields of ASCII log file.

Parameters

- **data** (*Model*, optional) – Log record.
- **name** (*Optional[str]*) – Log file name.

Returns The converted log string.

Return type str

dump_tail()
Serialise trailing fields of ASCII log file.

Returns The converted log string.

Return type str

zlogging.dumper.**write_json**(*data*, *filename*, *writer=None*, **args*, ***kwargs*)
Write JSON log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **filename** (*os.PathLike*) – Log file name.
- **writer** (*JSONWriter*, optional) – Writer class.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

zlogging.dumper.**dump_json**(*data*, *file*, *writer=None*, **args*, ***kwargs*)
Write JSON log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **writer** (*JSONWriter*, optional) – Writer class.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

zlogging.dumper.**dumps_json**(*data=None*, *writer=None*, **args*, ***kwargs*)
Write JSON log string.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **writer** (*JSONWriter*, optional) – Writer class.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The JSON log string.

Return type str

zlogging.dumper.**write_ascii**(*data*, *filename*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Write ASCII log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **filename** (*os.PathLike*) – Log file name.
- **writer** (*ASCIIWriter*, optional) – Writer class.
- **separator** (str or bytes, optional) – Field separator when writing log lines.
- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

zlogging.dumper.**dump_ascii**(*data*, *file*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Write ASCII log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **writer** (*ASCIIWriter*, optional) – Writer class.
- **separator** (str or bytes, optional) – Field separator when writing log lines.
- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

zlogging.dumper.**dumps_ascii**(*data=None*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Write ASCII log string.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **writer** (*ASCIIWriter*, optional) – Writer class.
- **separator** (str or bytes, optional) – Field separator when writing log lines.
- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The JSON log string.

Return type str

zlogging.dumper.**write**(*data*, *filename*, *format*, **args*, ***kwargs*)

Write Bro/Zeek log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **filename** (*os.PathLike*) – Log file name.
- **format** (*str*) – Log format.
- ***args** – See *write_json()* and *write_ascii()* for more information.
- ****kwargs** – See *write_json()* and *write_ascii()* for more information.

Raises ***WriterFormatError*** – If format is not supported.

zlogging.dumper.**dump**(*data*, *file*, *format*, **args*, ***kwargs*)
Write Bro/Zeek log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **format** (*str*) – Log format.
- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- ***args** – See *dump_json()* and *dump_ascii()* for more information.
- ****kwargs** – See *dump_json()* and *dump_ascii()* for more information.

Raises ***WriterFormatError*** – If format is not supported.

zlogging.dumper.**dumps**(*data*, *format*, **args*, ***kwargs*)
Write Bro/Zeek log string.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **format** (*str*) – Log format.
- ***args** – See *dumps_json()* and *dumps_ascii()* for more information.
- ****kwargs** – See *dumps_json()* and *dumps_ascii()* for more information.

Raises ***WriterFormatError*** – If format is not supported.

Abstract Base Dumpers

class zlogging.dumper.**BaseWriter**
Bases: object

Basic log writer.

abstract property format
str: Log file format.

write(*filename*, *data*)
Write log file.

Parameters

- **filename** (*os.PathLike*) – Log file name.
- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The file offset after writing.

Return type int

abstract write_file(*file*, *data*)
Write log file.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The file offset after writing.

Return type int

abstract write_line(*file*, *data*, *lineno=0*)
Write log line as one-line record.

Parameters

- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The file offset after writing.

Return type int

abstract dump_file(*data*)
Serialise records to a log line.

Parameters **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The converted log string.

Return type str

abstract dump_line(*data*, *lineno=0*)
Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The converted log string.

Return type str

dump(*data*, *file*)
Write log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.

Returns The file offset after writing.

Return type int

dumps(*data*)
Serialise records to a log line.

Parameters **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.

Returns The converted log string.

Return type str

1.1.2 Loaders

Predefined Loaders

Bro/Zeek log loader.

class zlogging.loader.**JSONParser**(*model=None*)
Bases: *zlogging.loader.BaseParser*

JSON log parser.

Parameters **model** (*Model* class, optional) – Field declrations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.

Variables **model** (*Model* class, optional) – Field declrations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.

Warns **JSONParserWarning** – If `model` is not specified.

property format
str: Log file format.

parse_file(*file*)
Parse log file.

Parameters **file** (*_io.BufferedReader*) – Log file object opened in binary mode.

Returns

The parsed log as a *Model* per line.

Return type *JSONInfo*

parse_line(*line*, *lineno=0*)
Parse log line as one-line record.

Parameters

- **line** (*bytes*) – A simple line of log.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The parsed log as a plain `dict`.

Raises ***JSONParserError*** – If failed to serialise the `line` from JSON.

Return type Dict[str, Any]

class zlogging.loader.**ASCIIParser**(*type_hook=None*, *enum_namespaces=None*, *bare=False*)
Bases: *zlogging.loader.BaseParser*

ASCII log parser.

Parameters

- **type_hook** (`dict` mapping `str` and *BaseType* class, optional) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.

Variables

- **__type__** (dict mapping str and *BaseType* class) – Bro/Zeek type parser hooks.
- **enum_namespaces** (List[str]) – Namespaces to be loaded.
- **bare** (*bool*) – If True, do not load zeek namespace by default.

property format
str: Log file format.

parse_file(*file*)
Parse log file.

Parameters **file** (*_io.BufferedReader*) – Log file object opened in binary mode.

Returns

The parsed log as a *Model* per line.

Return type *ASCIIInfo*

Warns **ASCIIParserWarning** – If the ASCII log file exited with error, see *ASCIIInfo.exit_with_error* for more information.

parse_line(*line*, *lineno=0*, *separator=b'\t'*, *parser=None*)
Parse log line as one-line record.

Parameters

- **line** (*bytes*) – A simple line of log.
- **lineno** (*Optional[int]*) – Line number of current line.
- **separator** (*Optional[bytes]*) – Data separator.
- **parser** (List of *BaseType*, required) – Field data type parsers.

Returns The parsed log as a plain dict.

Raises ***ASCIIPaserError*** – If parser is not provided; or failed to serialise line as ASCII.

Return type Dict[str, Any]

zlogging.loader.**parse_json**(*filename*, *parser=None*, *model=None*, **args*, ***kwargs*)
Parse JSON log file.

Parameters

- **filename** (*os.PathLike*) – Log file name.
- **parser** (*JSONParser*, optional) – Parser class.
- **model** (Model class, optional) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed JSON log data.

Return type *zlogging._data.JSONInfo*

zlogging.loader.**load_json**(*file*, *parser=None*, *model=None*, **args*, ***kwargs*)
Parse JSON log file.

Parameters

- **file** (*_io.BufferedReader*) – Log file object opened in binary mode.

- **parser** (*JSONParser*, optional) – Parser class.
- **model** (`Model` class, optional) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed JSON log data.

Return type *zlogging._data.JSONInfo*

`zlogging.loader.`**loads_json**(*data*, *parser=None*, *model=None*, **args*, ***kwargs*)

Parse JSON log string.

Parameters

- **data** (*AnyStr*) – Log string as binary or encoded string.
- **parser** (*JSONParser*, optional) – Parser class.
- **model** (`Model` class, optional) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed JSON log data.

Return type *zlogging._data.JSONInfo*

`zlogging.loader.`**parse_ascii**(*filename*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log file.

Parameters

- **filename** (*os.PathLike*) – Log file name.
- **parser** (*ASCIIParser*, optional) – Parser class.
- **type_hook** (`dict` mapping `str` and *BaseType* class, optional) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed ASCII log data.

Return type *zlogging._data.ASCIIInfo*

`zlogging.loader.`**load_ascii**(*file*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log file.

Parameters

- **file** (*_io.BufferedReader*) – Log file object opened in binary mode.
- **parser** (*ASCIIParser*, optional) – Parser class.

- **type_hook** (`dict` mapping `str` and *BaseType* class, optional) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed ASCII log data.

Return type *zlogging._data.ASCIIInfo*

`zlogging.loader.`**`loads_ascii`**(*data*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log string.

Parameters

- **data** (*AnyStr*) – Log string as binary or encoded string.
- **parser** (*ASCIIParser*, optional) – Parser class.
- **type_hook** (`dict` mapping `str` and *BaseType* class, optional) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns The parsed ASCII log data.

Return type *zlogging._data.ASCIIInfo*

`zlogging.loader.`**`parse`**(*filename*, **args*, ***kwargs*)

Parse Bro/Zeek log file.

Parameters

- **filename** (*os.PathLike*) – Log file name.
- ***args** – See *parse_json()* and *parse_ascii()* for more information.
- ****kwargs** – See *parse_json()* and *parse_ascii()* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

`zlogging.loader.`**`load`**(*file*, **args*, ***kwargs*)

Parse Bro/Zeek log file.

Parameters

- **file** (*_io.BufferedReader*) – Log file object opened in binary mode.
- ***args** – See *load_json()* and *load_ascii()* for more information.
- ****kwargs** – See *load_json()* and *load_ascii()* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

`zlogging.loader.`**`loads`**(*data*, **args*, ***kwargs*)
Parse Bro/Zeek log string.

Parameters

- **`data`** (*`AnyStr`*) – Log string as binary or encoded string.
- **`*args`** – See *`loads_json()`* and *`loads_ascii()`* for more information.
- **`**kwargs`** – See *`loads_json()`* and *`loads_ascii()`* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

Abstract Base Loaders

`class` `zlogging.loader.`**`BaseParser`**
Bases: `object`

Basic log parser.

`abstract property format`
str: Log file format.

`parse`(*filename*)
Parse log file.

Parameters **`filename`** (*`os.PathLike`*) – Log file name.

Returns The parsed log as an *`ASCIIInfo`* or *`JSONInfo`*.

Return type *zlogging._data.Info*

`abstract parse_file`(*file*)
Parse log file.

Parameters **`file`** (*`_io.BufferedReader`*) – Log file object opened in binary mode.

Returns The parsed log as a *`Model`* per line.

Return type *`Info`*

`abstract parse_line`(*line*, *lineno=0*)
Parse log line as one-line record.

Parameters

- **`line`** (*`bytes`*) – A simple line of log.
- **`lineno`** (*`Optional[int]`*) – Line number of current line.

Returns The parsed log as a plain `dict`.

Return type Dict[str, Any]

`load`(*file*)
Parse log file.

Parameters **`file`** (*`_io.BufferedReader`*) – Log file object opened in binary mode.

Returns The parsed log as a *Model* per line.

Return type *Info*

loads(*line*, *lineno=0*)
Parse log line as one-line record.

Parameters

- **line** (*bytes*) – A simple line of log.
- **lineno** (*Optional[int]*) – Line number of current line.

Returns The parsed log as a plain `dict`.

Return type Dict[str, Any]

1.1.3 Data Model

Bro/Zeek log data model.

class zlogging.model.**Model**(**args*, ***kwargs*)
Bases: `object`

Log data model.

Variables

- **__fields__** (`OrderedDict` mapping `str` and *BaseType*) – Fields of the data model.
- **__record_fields__** (`OrderedDict` mapping `str` and `RecordType`) – Fields of `record` data type in the data model.
- **__empty_field__** (*bytes*) – Placeholder for empty field.
- **__unset_field__** (*bytes*) – Placeholder for unset field.
- **__set_separator__** (*bytes*) – Separator for set/vector fields.

Warns **BroDeprecationWarning** – Use of `bro_*` type annotations.

Raises

- ***ModelValueError*** – In case of inconsistency between field data types, or values of `unset_field`, `empty_field` and `set_separator`.
- ***ModelTypeError*** – Wrong parameters when initialisation.

Note: Customise the `Model.__post_init__` method in your subclassed data model to implement your own ideas.

Example

Define a custom log data model using the prefines Bro/Zeek data types, or subclasses of *BaseType*:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

Or you may use type annotations as PEP 484 introduced when declaring data models. All available type hints can be found in `typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

However, when mixing annotations and direct assignments, annotations will take proceedings, i.e. the *Model* class shall process first annotations then assignments. Should there be any conflicts, `ModelError` will be raised.

See also:

See `_aux_expand_typing()` for more information about processing the fields.

property fields
`OrderedDict` mapping `str` and *BaseType*: fields of the data model

property unset_field
bytes: placeholder for empty field

property empty_field
bytes: placeholder for unset field

property set_separator
bytes: separator for set/vector fields

__post_init__()
Post-processing customisation.

__call__(*format*)
Serialise data model with given format.

Parameters **format** (*str*) – Serialisation format.

Returns The serialised data.

Raises ***ModelFormatError*** – If `format` is not supproted, i.e. `Mode.to{format}()` does not exist.

Return type Any

tojson()
Serialise data model as JSON log format.

Returns An `OrderedDict` mapping each field and serialised JSON serialisable data.

Return type OrderedDict[str, Any]

toascii()
Serialise data model as ASCII log format.

Returns An `OrderedDict` mapping each field and serialised text data.

Return type OrderedDict[str, str]

asdict(*dict_factory=None*)
Convert data model as a dictionary mapping field names to field values.

Parameters **dict_factory** (*Optional[type]*) – If given, `dict_factory` will be used instead of built-in `dict`.

Returns A dictionary mapping field names to field values.

Return type Dict[str, Any]

astuple(*tuple_factory=None*)
Convert data model as a tuple of field values.

Parameters **tuple_factory** (*Optional[type]*) – If given, `tuple_factory` will be used instead of built-in `tuple`.

Returns A tuple of field values.

Return type Tuple[Any]

`zlogging.model.`**new_model**(*name*, ***fields*)
Create a data model dynamically with the appropriate fields.

Parameters

- **name** (*str*) – data model name
- ****fields** – defined fields of the data model
- **Any] fields** (*Dict[str,*) –

Returns created data model

Return type *Model*

Examples

Typically, we define a data model by subclassing the *Model* class, as following:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

when defining dynamically with *new_model()*, the definition above can be rewrote to:

```
MyLog = new_model('MyLog', field_one=StringType(), field_two=SetType(element_
↪type=PortType))
```

1.1.4 Data Types

Bro/Zeek Types

Bro/Zeek data types.

class `zlogging.types.`**BoolType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `bool` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, bool]*) – raw data

Returns The parsed boolean data. If `data` is *unset*, `None` will be returned.

Raises ***ZeekValueError*** – If `data` is NOT *unset* and NOT `T` (`True`) nor `F` (`False`) in Bro/Zeek script language.

Return type Union[None, bool]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, bool]*) – raw data

Returns The JSON serialisable boolean data.

Return type Union[None, bool]

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, bool]*) – raw data

Returns `T` if `True`, `F` if `False`.

Return type str

class zlogging.types.**CountType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `count` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_ulong]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_ulong]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_ulong]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_ulong]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.types.**IntType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `int` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_long]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_long]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_long]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_long]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.types.**DoubleType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `double` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, decimal.Decimal]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, decimal.Decimal]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, decimal.Decimal]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, decimal.Decimal]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.types.**TimeType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `time` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, datetime.datetime]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, datetime.datetime]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, datetime.datetime]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, datetime.datetime]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.types.**IntervalType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `interval` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, datetime.timedelta]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, datetime.timedelta]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, datetime.timedelta]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, datetime.timedelta]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.types.**StringType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `string` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.

- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for set/vector fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse data from string.

Parameters **data** (*Union[AnyStr, memoryview, bytearray]*) – raw data

Returns The parsed string data. If data is *unset*, None will be returned.

Return type Union[None, ByteString]

tojson(*data*)
Serialize data as JSON log format.

Parameters **data** (*Union[None, ByteString]*) – raw data

Returns The JSON serialisable string data encoded in ASCII.

Return type str

toascii(*data*)
Serialize data as ASCII log format.

Parameters **data** (*Union[None, ByteString]*) – raw data

Returns The ASCII encoded string data.

Return type str

class zlogging.types.**AddrType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek addr data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.

- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The parsed IP address. If `data` is *unset*, `None` will be returned.

Return type Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The JSON serialisable IP address string.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The ASCII representation of the IP address.

Return type str

class zlogging.types.**PortType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `port` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_ushort]*) – raw data

Returns The parsed port number. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_ushort]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_ushort]*) – raw data

Returns The JSON serialisable port number string.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_ushort]*) – raw data

Returns The ASCII representation of the port number.

Return type str

class zlogging.types.**SubnetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `subnet` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The parsed IP network. If `data` is *unset*, `None` will be returned.

Return type Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The JSON serialisable IP network string.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The ASCII representation of the IP network.

Return type str

class zlogging.types.**EnumType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *namespaces=None*, *bare=False*, *enum_hook=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `enum` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- **namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.
- **enum_hook** (`dict` mapping of `str` and `enum.Enum`, optional) – Additional enum to be included in the namespace.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **enum_namespaces** (`dict` mapping `str` and `enum.Enum`) – Global namespace for `enum` data type.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, enum.Enum]*) – raw data

Returns The parsed enum data. If `data` is *unset*, `None` will be returned.

Warns **ZeekValueWarning** – If `date` is not defined in the enum namespace.

Return type Union[None, enum.Enum]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, enum.Enum]*) – raw data

Returns The JSON serialisable enum data.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, enum.Enum]*) – raw data

Returns The ASCII representation of the enum data.

Return type str

class zlogging.types.**SetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)
Bases: *zlogging.types._GenericType*, typing.Generic

Bro/Zeek `set` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If `element_type` is not supplied.
- ***ZeekValueError*** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
>>> SetType[StringType]
```

which is the same **at runtime** as following:

```
>>> SetType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *`_SimpleType`*.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*`Union[AnyStr, Set[data]]`*) – raw data

Returns The parsed set data. If `data` is *unset*, `None` will be returned.

Return type Union[None, Set[data]]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*`Union[None, Set[data]]`*) – raw data

Returns The JSON serialisable set data.

Return type list

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*`Union[None, Set[data]]`*) – raw data

Returns The ASCII representation of the set data.

Return type str

class zlogging.types.**VectorType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)
Bases: *`zlogging.types._GenericType`*, `typing.Generic`

Bro/Zeek `vector` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- **element_type** (*`BaseType`* instance) – Data type of container's elements.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*`bytes`*) – Placeholder for empty field.
- **unset_field** (*`bytes`*) – Placeholder for unset field.

- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If `element_type` is not supplied.
- ***ZeekValueError*** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
>>> VectorType[StringType]
```

which is the same **at runtime** as following:

```
>>> VectorType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *_SimpleType*.

---

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, List[data]]*) – raw data

Returns The parsed list data. If `data` is *unset*, `None` will be returned.

Return type Union[None, List[data]]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, List[data]]*) – raw data

Returns The JSON serialisable list data.

Return type list

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, List[data]]*) – raw data

Returns The ASCII representation of the list data.

Return type str

class zlogging.types.**RecordType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***element_mapping*)
Bases: *zlogging.types._VariadicType*

Bro/Zeek `record` data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – element_mapping (dict mapping str and *BaseType* instance): Data type of container's elements.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for set/vector fields.
- **element_mapping** (dict mapping str and *BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If element_mapping is not supplied.
- ***ZeekValueError*** – If element_mapping is not a valid Bro/Zeek data type; or in case of inconsistency from empty_field, unset_field and set_separator of each field.

Note: A valid element_mapping should be a *simple* or *generic* data type, i.e. a subclass of *_SimpleType* or *_GenericType*.

See also:

See _aux_expand_typing() for more information about processing the fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

Abstract Base Types

class zlogging.types.**BaseType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: object

Base Bro/Zeek data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

abstract property python_type
type: Corresponding Python type annotation.

abstract property zeek_type
str: Corresponding Zeek type name.

property bro_type
str: Corresponding Bro type name.

__call__(*data*)
Parse `data` from string.

Parameters **data** (*Any*) –

Return type Any

abstract parse(*data*)
Parse `data` from string.

Parameters **data** (*Any*) –

Return type Any

abstract tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Any*) –

Return type Any

abstract toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Any*) –

Return type str

class zlogging.types.**_SimpleType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types.BaseType*

Simple data type.

In Bro/Zeek script language, such simple type includes `bool`, `count`, `int`, `double`, `time`, `interval`, `string`, `addr`, `port`, `subnet` and `enum`.

To support arbitrary typing as required in *JSONParser*, `any`, the arbitrary date type is also included.

Parameters

- **empty_field** (*Optional[AnyStr]*) –
- **unset_field** (*Optional[AnyStr]*) –
- **set_separator** (*Optional[AnyStr]*) –

class zlogging.types.**_GenericType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types.BaseType*

Generic data type.

In Bro/Zeek script language, such generic type includes `set` and `vector`, which are also known as *container* types.

Parameters

- **empty_field** (*Optional[AnyStr]*) –
- **unset_field** (*Optional[AnyStr]*) –
- **set_separator** (*Optional[AnyStr]*) –

class zlogging.types.**_VariadicType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types.BaseType*

Variadic data type.

In Bro/Zeek script language, such variadic type refers to `record`, which is also a *container* type.

Parameters

- **empty_field** (*Optional[AnyStr]*) –
- **unset_field** (*Optional[AnyStr]*) –
- **set_separator** (*Optional[AnyStr]*) –

parse(*data*)

Not supported for a variadic data type.

Parameters **data** (*Any*) – data to process

Raises ***ZeekNotImplemented*** – If try to call such method.

Return type NoReturn

tojson(*data*)

Not supported for a variadic data type.

Parameters **data** (*Any*) – data to process

Raises ***ZeekNotImplemented*** – If try to call such method.

Return type NoReturn

toascii(*data*)

Not supported for a variadic data type.

Parameters **data** (*Any*) – data to process

Raises ***ZeekNotImplemented*** – If try to call such method.

Return type NoReturn

Internal Data

zlogging.types.**_data = ~data**

A typing variable representing all valid data types for Bro/Zeek log framework.

Type type

1.1.5 Typing Annotations

Zeek Data Types

zlogging.typing.**zeek_addr = ~addr**
Zeek addr data type.

Type type

zlogging.typing.**zeek_bool = ~bool**
Zeek bool data type.

Type type

zlogging.typing.**zeek_count = ~count**
Zeek count data type.

Type type

zlogging.typing.**zeek_double = ~double**
Zeek double data type.

Type type

zlogging.typing.**zeek_enum = ~enum**
Zeek enum data type.

Type type

zlogging.typing.**zeek_interval = ~interval**
Zeek interval data type.

Type type

zlogging.typing.**zeek_int = ~int**
Zeek int data type.

Type type

zlogging.typing.**zeek_port = ~port**
Zeek port data type.

Type type

zlogging.typing.**zeek_record = ~record**
Zeek record data type.

Note: As a *variadic* data type, it supports the typing proxy as `TypedDict`, introduced in PEP 589:

```
class MyLog(zeek_record):
    field_one: zeek_int
    field_two: zeek_set[zeek_port]
```

which is the same **at runtime** as following:

```
RecordType(field_one=IntType,
           field_two=SetType(element_type=PortType))
```

See also:

See *expand_typing()* for more information about the processing of typing proxy.

zlogging.typing.**zeek_set = ~set**
Zeek `set` data type.

Note: As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
class MyLog(zeek_record):
    field_one: zeek_set[zeek_str]
```

which is the same **at runtime** as following:

```
class MyLog(zeek_record):
    field_one = SetType(element_type=StringType())
```

---

zlogging.typing.**zeek_string = ~string**
Zeek `string` data type.

Type type

zlogging.typing.**zeek_subnet = ~subnet**
Zeek `subnet` data type.

Type type

zlogging.typing.**zeek_time = ~time**
Zeek `time` data type.

Type type

zlogging.typing.**zeek_vector = ~vector**
Zeek `vector` data type.

Note: As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
class MyLog(zeek_record):
    field_one: zeek_vector[zeek_str]
```

which is the same **at runtime** as following:

```
class MyLog(zeek_record):
    field_one = VectorType(element_type=StringType())
```

Bro Data Types

Use of `bro` is deprecated. Please use `zeek` instead.

zlogging.typing.**bro_addr = ~bro_addr**
Bro `addr` data type.

Type type

zlogging.typing.**bro_bool = ~bro_bool**
Bro `bool` data type.

Type type

zlogging.typing.**bro_count = ~bro_count**
Bro `count` data type.

Type type

zlogging.typing.**bro_double = ~bro_double**
Bro `double` data type.

Type type

zlogging.typing.**bro_enum = ~bro_enum**
Bro `enum` data type.

Type type

zlogging.typing.**bro_interval = ~bro_interval**
Bro `interval` data type.

Type type

zlogging.typing.**bro_int = ~bro_int**
Bro `int` data type.

Type type

zlogging.typing.**bro_port = ~bro_port**
Bro `port` data type.

Type type

zlogging.typing.**bro_record = ~bro_record**
Bro `record` data type.

See also:

See *zeek_record* for more information.

zlogging.typing.**bro_set = ~bro_set**
Bro `set` data type.

See also:

See *zeek_set* for more information.

zlogging.typing.**bro_string = ~bro_string**
Bro `string` data type.

Type type

zlogging.typing.**bro_subnet = ~bro_subnet**
Bro `subnet` data type.

Type type

zlogging.typing.**bro_time = ~bro_time**
Bro `time` data type.

Type type

zlogging.typing.**bro_vector = ~bro_vector**
Bro `vector` data type.

See also:

See *zeek_vector* for more information.

1.1.6 Data Classes

Predefined Data Classes

Data classes for parsed logs.

class zlogging._data.**ASCIIInfo**(*path*, *open*, *close*, *data*, *exit_with_error*)
Bases: *zlogging._data.Info*

Parsed log info for ASCII logs.

The ASCII log will be stored as in this `dataclass`, as introduced in PEP 557.

Parameters

- **path** (`os.PathLike`) – The value is specified in the ASCII log file under `# path` directive.
- **open** (`datetime.datetime`) – The value is specified in the ASCII log file under `# open` directive.
- **close** (`datetime.datetime`) – The value is specified in the ASCII log file under `# close` directive.
- **data** (`list` or *Model*) – The log records parsed as a `list` of *Model* per line.
- **exit_with_error** (`bool`) – When exit with error, the ASCII log file doesn't has a `# close` directive.

Return type None

property format
str: Log file format.

path: typing.PathLike = None
Log path.

The value is specified in the ASCII log file under `# path` directive.

Type `os.PathLike`

open: typing.DateTime = None
Log open time.

The value is specified in the ASCII log file under `# open` directive.

Type `datetime.datetime`

close: typing.DateTime = None
Log close time.

The value is specified in the ASCII log file under `# close` directive.

Type `datetime.datetime`

data: typing.List[Model] = None
Log records.

The log records parsed as a `list` of *Model* per line.

Type `list` of *Model*

exit_with_error: bool = None
Log exit with error.

When exit with error, the ASCII log file doesn't has a `# close` directive.

Type bool

class zlogging._data.**JSONInfo**(*data*)
Bases: *zlogging._data.Info*

Parsed log info for JSON logs.

The JSON log will be stored as in this dataclass, as introduced in PEP 557.

Parameters **data** (list of *Model*) – The log records parsed as a list of *Model* per line.

Return type None

property format
str: Log file format.

data: typing.List[Model] = None
Log records.

The log records parsed as a list of *Model* per line.

Type list of *Model*

Abstract Base Data Classes

class zlogging._data.**Info**
Bases: object

Parsed log info.

The parsed log will be stored as in this dataclass, as introduced in PEP 557.

abstract property format
str: Log file format.

1.1.7 Exceptions & Warnings

Exceptions & warnings.

exception zlogging._exc.**ZeekException**
Bases: Exception

Base exception.

exception zlogging._exc.**ZeekWarning**
Bases: Warning

Base warning.

exception zlogging._exc.**ParserError**(*msg*, *lineno=None*, *field=None*)
Bases: *zlogging._exc.ZeekException*, ValueError

Error when parsing logs.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where parsing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** – (str) The field name where parsing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**JSONParserError**(*msg*, *lineno=None*, *field=None*)

Bases: *zlogging._exc.ParserError*, json.decoder.JSONDecodeError

Error when parsing JSON log.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where parsing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** – (str) The field name where parsing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**ASCIIPaserError**(*msg*, *lineno=None*, *field=None*)

Bases: *zlogging._exc.ParserError*

Error when parsing ASCII log.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where parsing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** – (str) The field name where parsing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**WriterError**(*msg*, *lineno=None*, *field=None*)

Bases: *zlogging._exc.ZeekException*, TypeError

Error when writing logs.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where writing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** (*str*) – The field name where writing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**JSONWriterError**(*msg*, *lineno=None*, *field=None*)
Bases: *zlogging._exc.WriterError*

Error when writing JSON logs.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where writing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** (*str*) – The field name where writing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**ASCIIWriterError**(*msg*, *lineno=None*, *field=None*)
Bases: *zlogging._exc.WriterError*

Error when writing ASCII logs.

Parameters

- **msg** (*str*) – The unformatted error message.
- **lineno** (int, optional) – The line corresponding to the failure.
- **field** (str, optional) – The field name where writing failed.

Variables

- **msg** (*str*) – The unformatted error message.
- **field** (*str*) – The field name where writing failed.
- **lineno** (*int*) – The line corresponding to the failure.

exception zlogging._exc.**WriterFormatError**(*msg*, *lineno=None*, *field=None*)
Bases: *zlogging._exc.WriterError*, ValueError

Unsupported format.

Parameters

- **msg** (*str*) – the unformatted error message
- **lineno** (int, optional) – the line corresponding to the failure
- **field** (str, optional) – the field name where writing failed

Variables

- **msg** (*str*) – the unformatted error message
- **field** (*str*) – the field name where writing failed
- **lineno** (*int*) – the line corresponding to the failure

exception zlogging._exc.**ParserWarning**
Bases: *zlogging._exc.ZeekWarning*, UserWarning

Warning when parsing logs.

exception zlogging._exc.**JSONParserWarning**
Bases: *zlogging._exc.ParserWarning*

Warning when parsing logs in JSON format.

exception zlogging._exc.**ASCIIParserWarning**
Bases: *zlogging._exc.ParserWarning*

Warning when parsing logs in ASCII format.

exception zlogging._exc.**ZeekTypeError**
Bases: *zlogging._exc.ZeekException*, TypeError

Invalid Bro/Zeek data type.

exception zlogging._exc.**ZeekValueError**
Bases: *zlogging._exc.ZeekException*, ValueError

Invalid Bro/Zeek data value.

exception zlogging._exc.**ZeekNotImplemented**
Bases: *zlogging._exc.ZeekException*, NotImplementedError

Method not implemented.

exception zlogging._exc.**ModelError**
Bases: *zlogging._exc.ZeekException*

Invalid model data.

exception zlogging._exc.**ModelTypeError**
Bases: *zlogging._exc.ModelError*, TypeError

Invalid model data type.

exception zlogging._exc.**ModelValueError**
Bases: *zlogging._exc.ModelError*, ValueError

Invalid model data value.

exception zlogging._exc.**ModelFormatError**
Bases: *zlogging._exc.ModelError*, ValueError

Unsupported format.

exception zlogging._exc.**ZeekValueWarning**
Bases: *zlogging._exc.ZeekWarning*, UserWarning

Dubious Bro/Zeek data value.

exception zlogging._exc.**BroDeprecationWarning**
Bases: *zlogging._exc.ZeekWarning*, DeprecationWarning

Bro is now deprecated, use Zeek instead.

1.1.8 Internal Auxiliary Functions

Auxiliary functions.

zlogging._aux.**readline**(*file*, *separator=b'\t'*, *maxsplit=-1*, *decode=False*)
Wrapper for `file.readline()` function.

Parameters

- **file** (*_io.BufferedReader*) – Log file object opened in binary mode.
- **separator** (*bytes*) – Data separator.
- **maxsplit** (*int*) – Maximum number of splits to do; see `bytes.split()` and `str.split()` for more information.
- **decode** (*bool*) – If decide the buffered string with `ascii` encoding.

Returns The splitted line as a `list` of `bytes`, or as `str` if `decode` if set to `True`.

Return type List[AnyStr]

zlogging._aux.**decimal_toascii**(*data*, *infinite=None*)
Convert `decimal.Decimal` to ASCII.

Parameters

- **data** (*decimal.Decimal*) – A `decimal.Decimal` object.
- **infinite** (*str*) – The ASCII representation of infinite numbers (`NaN` and infinity).

Returns The converted ASCII string.

Return type str

Example

When converting a `decimal.Decimal` object, for example:

```
>>> d = decimal.Decimal('-123.123456789')
```

the function will preserve only **6 digits** of its fractional part, i.e.:

```
>>> decimal_toascii(d)
'-123.123456'
```

Note: Infinite numbers, i.e. `NaN` and infinity (`inf`), will be converted as the value specified in `infinite`, in default the string representation of the number itself, i.e.:

- `NaN` -> `'NaN'`
- Infinity -> `'Infinity'`

---

zlogging._aux.**float_toascii**(*data*, *infinite=None*)
Convert `float` to ASCII.

Parameters

- **data** (*float*) – A `float` number.
- **infinite** (*str*) – The ASCII representation of infinite numbers (`NaN` and infinity).

Returns The converted ASCII string.

Return type str

Example

When converting a `float` number, for example:

```
>>> f = -123.123456789
```

the function will preserve only **6 digits** of its fractional part, i.e.:

```
>>> float_toascii(f)
'-123.123456'
```

Note: Infinite numbers, i.e. `NaN` and infinity (`inf`), will be converted as the value specified in `infinite`, in default the string representation of the number itself, i.e.:

- `NaN` -> `'nan'`
- Infinity -> `'inf'`

`zlogging._aux.`**`unicode_escape`**(*string*)

Conterprocess of `bytes.decode('unicode_escape')()`.

Parameters **`string`** (*bytes*) – The bytestring to be escaped.

Returns The escaped bytestring as an encoded string

Return type str

Example

```
>>> b'\x09'.decode('unicode_escape')
'\t'
>>> unicode_escape(b'\t')
'\x09'
```

`zlogging._aux.`**`expand_typing`**(*cls*, *exc=None*)

Expand typing annotations.

Parameters

- **`cls`** (*`Model`* or *`RecordType`* object) – a variadic class which supports PEP 484 style attribute typing annotations
- **`exc`** (*`Optional[ValueError]`*) – (`ValueError`, optional): exception to be used in case of inconsistent values for `unset_field`, `empty_field` and `set_separator`

Returns

The returned dictionary contains the following directives:

- **`fields` (`OrderedDict` mapping `str` and *`BaseType`*):** a mapping proxy of field names and their corresponding data types, i.e. an instance of a *`BaseType`* subclass
- **`record_fields` (`OrderedDict` mapping `str` and *`RecordType`*):** a mapping proxy for fields of `record` data type, i.e. an instance of *`RecordType`*

- `unset_fields` (`bytes`): placeholder for unset field
- `empty_fields` (`bytes`): placeholder for empty field
- `set_separator` (`bytes`): separator for `set`/`vector` fields

Return type `Dict[str, Any]`

Warns **BroDeprecationWarning** – Use of `bro_*` prefixed typing annotations.

Raises **ValueError** – In case of inconsistent values for `unset_field`, `empty_field` and `set_separator`.

Example

Define a custom log data model from *Model* using the prefines Bro/Zeek data types, or subclasses of *BaseType*:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

Or you may use type annotations as PEP 484 introduced when declaring data models. All available type hints can be found in `zlogging.typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

However, when mixing annotations and direct assignments, annotations will take proceedings, i.e. the function shall process first typing annotations then `cls` attribute assignments. Should there be any conflicts, the `exc` will be raised.

Note: Fields of *zlogging.types.RecordType* type will be expanded as plain fields of the `cls`, i.e. for the variadic class as below:

```
class MyLog(Model):
    record = RecrodType(one=StringType(),
                        two=VectorType(element_type=CountType()))
```

will have the following fields:

- `record.one` -> `string` data type
- `record.two` -> `vector[count]` data type

1.1.9 Enum Namespace

Module Contents

Bro/Zeek enum namespace.

zlogging.enum.**globals**(**namespaces*, *bare=False*)

Generate Bro/Zeek `enum` namespace.

Parameters

- ***namespaces** – Namespaces to be loaded.
- **bare** (*bool*) – If `True`, do not load `zeek` namespace by default.

Keyword Arguments **bare** – If `True`, do not load `zeek` namespace by default.

Returns Global enum namespace.

Return type `dict` mapping of `str` and `Enum`

Warns **BroDeprecationWarning** – If `bro` namespace used.

Raises **ValueError** – If `namespace` is not defined.

Note: For back-port compatibility, the `bro` namespace is an alias of the `zeek` namespace.

Namespaces

`Broker` Namespace

Namespace: `Broker`.

class zlogging.enum.Broker.**DataType**

Bases: `enum.IntFlag`

Enumerates the possible types that Broker::Data may be in terms of Zeek data types.

c.f. base/bif/data.bif.zeek

ADDR = 64

BOOL = 2

COUNT = 8

DOUBLE = 16

ENUM = 2048

INT = 4

INTERVAL = 1024

NONE = 1

PORT = 256

SET = 4096

STRING = 32

SUBNET = 128

TABLE = 8192

TIME = 512

VECTOR = 16384

class zlogging.enum.Broker.**Type**
Bases: enum.IntFlag

The type of a Broker activity being logged.

c.f. base/frameworks/broker/log.zeek

ERROR = 2

STATUS = 1

class zlogging.enum.Broker.**ErrorCode**
Bases: enum.IntFlag

Enumerates the possible error types.

c.f. base/frameworks/broker/main.zeek

BACKEND_FAILURE = 2048

CAF_ERROR = 8192

INVALID_DATA = 1024

MASTER_EXISTS = 32

NO_SUCH_KEY = 128

NO_SUCH_MASTER = 64

PEER_INCOMPATIBLE = 2

PEER_INVALID = 4

PEER_TIMEOUT = 16

PEER_UNAVAILABLE = 8

REQUEST_TIMEOUT = 256

STALE_DATA = 4096

TYPE_CLASH = 512

UNSPECIFIED = 1

class zlogging.enum.Broker.**PeerStatus**
Bases: enum.IntFlag

The possible states of a peer endpoint.

c.f. base/frameworks/broker/main.zeek

CONNECTED = 4

CONNECTING = 2

DISCONNECTED = 16

INITIALIZING = 1

PEERED = 8

RECONNECTING = 32

class zlogging.enum.Broker.**BackendType**
Bases: enum.IntFlag

Enumerates the possible storage backends.

c.f. base/frameworks/broker/store.zeek

MEMORY = 1

ROCKSDB = 4

SQLITE = 2

class zlogging.enum.Broker.**QueryStatus**
Bases: enum.IntFlag

Whether a data store query could be completed or not.

c.f. base/frameworks/broker/store.zeek

FAILURE = 2

SUCCESS = 1

Cluster Namespace

Namespace: Cluster.

class zlogging.enum.Cluster.**NodeType**
Bases: enum.IntFlag

Types of nodes that are allowed to participate in the cluster configuration.

c.f. base/frameworks/cluster/main.zeek

CONTROL = 2

LOGGER = 4

MANAGER = 8

NONE = 1

PROXY = 16

TIME_MACHINE = 64

WORKER = 32

DCE_RPC Namespace

Namespace: DCE_RPC.

class zlogging.enum.DCE_RPC.**IfID**
Bases: enum.IntFlag

c.f. base/bif/plugins/Zeek_DCE_RPC.types.bif.zeek

ISCMActivator = 8192

drs = 512

epmapper = 2

lsa_ds = 8

lsarpc = 4

mgmt = 16

netlogon = 32

oxid = 4096

samr = 64

spoolss = 256

srvsvc = 128

unknown_if = 1

winspipe = 1024

wkssvc = 2048

class zlogging.enum.DCE_RPC.**PType**

Bases: enum.IntFlag

c.f. base/bif/plugins/Zeek_DCE_RPC.types.bif.zeek

ACK = 128

ALTER_CONTEXT = 16384

ALTER_CONTEXT_RESP = 32768

AUTH3 = 65536

BIND = 2048

BIND_ACK = 4096

BIND_NAK = 8192

CANCEL_ACK = 1024

CL_CANCEL = 256

CO_CANCEL = 262144

FACK = 512

FAULT = 8

NOCALL = 32

ORPHANED = 524288

PING = 2

REJECT = 64

REQUEST = 1

RESPONSE = 4

RTS = 1048576

SHUTDOWN = 131072

WORKING = 16

`HTTP` Namespace

Namespace: `HTTP`.

class `zlogging.enum.HTTP.`**Tags**
Bases: `enum.IntFlag`

Indicate a type of attack or compromise in the record to be logged.

c.f. base/protocols/http/main.zeek

COOKIE_SQLI = 8

EMPTY = 1

POST_SQLI = 4

URI_SQLI = 2

`Input` Namespace

Namespace: `Input`.

class `zlogging.enum.Input.`**Event**
Bases: `enum.IntFlag`

Type that describes what kind of change occurred.

c.f. base/frameworks/input/main.zeek

EVENT_CHANGED = 2

EVENT_NEW = 1

EVENT_REMOVED = 4

class `zlogging.enum.Input.`**Mode**
Bases: `enum.IntFlag`

Type that defines the input stream read mode.

c.f. base/frameworks/input/main.zeek

MANUAL = 1

REREAD = 2

STREAM = 4

class `zlogging.enum.Input.`**Reader**
Bases: `enum.IntFlag`

c.f. base/frameworks/input/main.zeek

READER_ASCII = 1

READER_BENCHMARK = 2

READER_BINARY = 4

READER_CONFIG = 8

READER_RAW = 16

READER_SQLITE = 32

`Intel` Namespace

Namespace: `Intel`.

class `zlogging.enum.Intel.`**Type**
Bases: `enum.IntFlag`

Enum type to represent various types of intelligence data.

c.f. base/frameworks/intel/main.zeek

ADDR = 1

CERT_HASH = 128

DOMAIN = 32

EMAIL = 16

FILE_HASH = 512

FILE_NAME = 1024

PUBKEY_HASH = 256

SOFTWARE = 8

SUBNET = 2

URL = 4

USER_NAME = 64

class `zlogging.enum.Intel.`**Where**
Bases: `enum.IntFlag`

Enum to represent where data came from when it was discovered. The convention is to prefix the name with IN_.

c.f. base/frameworks/intel/main.zeek

Conn__IN_ORIG = 2

Conn__IN_RESP = 4

DNS__IN_REQUEST = 32

DNS__IN_RESPONSE = 64

Files__IN_HASH = 8

Files__IN_NAME = 16

HTTP__IN_HOST_HEADER = 128

HTTP__IN_REFERRER_HEADER = 256

HTTP__IN_URL = 2048

HTTP__IN_USER_AGENT_HEADER = 512

HTTP__IN_X_FORWARDED_FOR_HEADER = 1024

IN_ANYWHERE = 1

SMB__IN_FILE_NAME = 33554432

SMTP__IN_CC = 65536

SMTP__IN_FROM = 16384

SMTP__IN_HEADER = 8388608

SMTP__IN_MAIL_FROM = 4096

SMTP__IN_MESSAGE = 1048576

SMTP__IN_RCPT_TO = 8192

SMTP__IN_RECEIVED_HEADER = 131072

SMTP__IN_REPLY_TO = 262144

SMTP__IN_TO = 32768

SMTP__IN_X_ORIGINATING_IP_HEADER = 524288

SSH__IN_SERVER_HOST_KEY = 2097152

SSH__SUCCESSFUL_LOGIN = 67108864

SSL__IN_SERVER_NAME = 4194304

X509__IN_CERT = 16777216

JSON Namespace

Namespace: JSON.

class zlogging.enum.JSON.**TimestampFormat**

Bases: enum.IntFlag

c.f. base/init-bare.zeek

TS_EPOCH = 1

TS_ISO8601 = 4

TS_MILLIS = 2

Known Namespace

Namespace: Known.

class zlogging.enum.Known.**ModbusDeviceType**

Bases: enum.IntFlag

c.f. policy/protocols/modbus/known-masters-slaves.zeek

MODBUS_MASTER = 1

MODBUS_SLAVE = 2

`LoadBalancing` Namespace

Namespace: `LoadBalancing`.

class `zlogging.enum.LoadBalancing.`**Method**
Bases: `enum.IntFlag`

c.f. policy/misc/load-balancing.zeek

AUTO_BPF = 1

`Log` Namespace

Namespace: `Log`.

class `zlogging.enum.Log.`**ID**
Bases: `enum.IntFlag`

Type that defines an ID unique to each log stream. Scripts creating new log streams need to redef this enum to add their own specific log ID. The log ID implicitly determines the default name of the generated log file.

c.f. base/frameworks/logging/main.zeek

Barnyard2_LOG = 1125899906842624

Broker_LOG = 2

CaptureLoss_LOG = 2251799813685248

Cluster_LOG = 16

Config_LOG = 8192

Conn_LOG = 524288

DCE_RPC_LOG = 1048576

DHCP_LOG = 2097152

DNP3_LOG = 4194304

DNS_LOG = 8388608

DPD_LOG = 256

FTP_LOG = 16777216

Files_LOG = 4

HTTP_LOG = 134217728

IRC_LOG = 268435456

Intel_LOG = 4096

KRB_LOG = 536870912

Known_CERTS_LOG = 18446744073709551616

Known_HOSTS_LOG = 72057594037927936

Known_MODBUS_LOG = 288230376151711744

Known_SERVICES_LOG = 144115188075855872

LoadedScripts_LOG = 9007199254740992

```
MQTT__CONNECT_LOG = 1152921504606846976
MQTT__PUBLISH_LOG = 4611686018427387904
MQTT__SUBSCRIBE_LOG = 2305843009213693952
Modbus__LOG = 1073741824
Modbus__REGISTER_CHANGE_LOG = 576460752303423488
NTLM__LOG = 4294967296
NTP__LOG = 8589934592
NetControl__CATCH_RELEASE = 140737488355328
NetControl__DROP = 131072
NetControl__LOG = 65536
NetControl__SHUNT = 262144
Notice__ALARM_LOG = 64
Notice__LOG = 32
OCSP__LOG = 562949953421312
OpenFlow__LOG = 32768
PE__LOG = 70368744177664
PacketFilter__LOG = 1024
RADIUS__LOG = 17179869184
RDP__LOG = 34359738368
RFB__LOG = 68719476736
Reporter__LOG = 8
SIP__LOG = 137438953472
SMB__AUTH_LOG = 549755813888
SMB__CMD_LOG = 9223372036854775808
SMB__FILES_LOG = 2199023255552
SMB__MAPPING_LOG = 1099511627776
SMTP__LOG = 4398046511104
SNMP__LOG = 274877906944
SOCKS__LOG = 8796093022208
SSH__LOG = 17592186044416
SSL__LOG = 33554432
Signatures__LOG = 512
Software__LOG = 2048
Stats__LOG = 18014398509481984
Syslog__LOG = 35184372088832
Traceroute__LOG = 4503599627370496
```

Tunnel__LOG = 16384

UNKNOWN = 1

Unified2__LOG = 281474976710656

WeirdStats__LOG = 36028797018963968

Weird__LOG = 128

X509__LOG = 67108864

ZeekygenExample__LOG = 36893488147419103232

mysql__LOG = 2147483648

class zlogging.enum.Log.**Writer**
Bases: enum.IntFlag

c.f. base/frameworks/logging/main.zeek

WRITER_ASCII = 1

WRITER_NONE = 2

WRITER_SQLITE = 4

MOUNT3 Namespace

Namespace: MOUNT3.

class zlogging.enum.MOUNT3.**auth_flavor_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

AUTH_DES = 8

AUTH_NULL = 1

AUTH_SHORT = 4

AUTH_UNIX = 2

class zlogging.enum.MOUNT3.**proc_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

PROC_DUMP = 4

PROC_END_OF_PROCS = 64

PROC_EXPORT = 32

PROC_MNT = 2

PROC_NULL = 1

PROC_UMNT = 8

PROC_UMNT_ALL = 16

class zlogging.enum.MOUNT3.**status_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

MNT3ERR_ACCES = 16

MNT3ERR_INVAL = 64

MNT3ERR_IO = 8

MNT3ERR_NAMETOOLONG = 128

MNT3ERR_NOENT = 4

MNT3ERR_NOTDIR = 32

MNT3ERR_NOTSUPP = 256

MNT3ERR_PERM = 2

MNT3ERR_SERVERFAULT = 512

MNT3_OK = 1

MOUNT3ERR_UNKNOWN = 1024

MQTT Namespace

Namespace: MQTT.

class zlogging.enum.MQTT.**SubUnsub**
Bases: enum.IntFlag

c.f. policy/protocols/mqtt/main.zeek

SUBSCRIBE = 1

UNSUBSCRIBE = 2

NFS3 Namespace

Namespace: NFS3.

class zlogging.enum.NFS3.**createmode_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

EXCLUSIVE = 4

GUARDED = 2

UNCHECKED = 1

class zlogging.enum.NFS3.**file_type_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

FTYPE_BLK = 4

FTYPE_CHR = 8

FTYPE_DIR = 2

FTYPE_FIFO = 64

FTYPE_LNK = 16

FTYPE_REG = 1

FTYPE_SOCK = 32

class zlogging.enum.NFS3.**proc_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

PROC_ACCESS = 16

PROC_COMMIT = 2097152

PROC_CREATE = 256

PROC_END_OF_PROCS = 4194304

PROC_FSINFO = 524288

PROC_FSSTAT = 262144

PROC_GETATTR = 2

PROC_LINK = 32768

PROC_LOOKUP = 8

PROC_MKDIR = 512

PROC_MKNOD = 2048

PROC_NULL = 1

PROC_PATHCONF = 1048576

PROC_READ = 64

PROC_READDIR = 65536

PROC_READDIRPLUS = 131072

PROC_READLINK = 32

PROC_REMOVE = 4096

PROC_RENAME = 16384

PROC_RMDIR = 8192

PROC_SETATTR = 4

PROC_SYMLINK = 1024

PROC_WRITE = 128

class zlogging.enum.NFS3.**stable_how_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

DATA_SYNC = 2

FILE_SYNC = 4

UNSTABLE = 1

class zlogging.enum.NFS3.**status_t**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

NFS3ERR_ACCES = 32

NFS3ERR_BADHANDLE = 2097152

NFS3ERR_BADTYPE = 134217728

NFS3ERR_BAD_COOKIE = 8388608

NFS3ERR_DQUOT = 262144

NFS3ERR_EXIST = 64

NFS3ERR_FBIG = 4096

NFS3ERR_INVAL = 2048

NFS3ERR_IO = 8

NFS3ERR_ISDIR = 1024

NFS3ERR_JUKEBOX = 268435456

NFS3ERR_MLINK = 32768

NFS3ERR_NAMETOOLONG = 65536

NFS3ERR_NODEV = 256

NFS3ERR_NOENT = 4

NFS3ERR_NOSPC = 8192

NFS3ERR_NOTDIR = 512

NFS3ERR_NOTEMPTY = 131072

NFS3ERR_NOTSUPP = 16777216

NFS3ERR_NOT_SYNC = 4194304

NFS3ERR_NXIO = 16

NFS3ERR_OK = 1

NFS3ERR_PERM = 2

NFS3ERR_REMOTE = 1048576

NFS3ERR_ROFS = 16384

NFS3ERR_SERVERFAULT = 67108864

NFS3ERR_STALE = 524288

NFS3ERR_TOOSMALL = 33554432

NFS3ERR_UNKNOWN = 536870912

NFS3ERR_XDEV = 128

class zlogging.enum.NFS3.**time_how_t**

Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

DONT_CHANGE = 1

SET_TO_CLIENT_TIME = 4

SET_TO_SERVER_TIME = 2

`NetControl` Namespace

Namespace: `NetControl`.

class `zlogging.enum.NetControl.`**InfoCategory**
Bases: `enum.IntFlag`

Type of an entry in the NetControl log.

c.f. base/frameworks/netcontrol/main.zeek

ERROR = 2

MESSAGE = 1

RULE = 4

class `zlogging.enum.NetControl.`**InfoState**
Bases: `enum.IntFlag`

State of an entry in the NetControl log.

c.f. base/frameworks/netcontrol/main.zeek

EXISTS = 4

FAILED = 8

REMOVED = 16

REQUESTED = 1

SUCCEEDED = 2

TIMEOUT = 32

class `zlogging.enum.NetControl.`**EntityType**
Bases: `enum.IntFlag`

Type defining the entity that a rule applies to.

c.f. base/frameworks/netcontrol/types.zeek

ADDRESS = 1

CONNECTION = 2

FLOW = 4

MAC = 8

class `zlogging.enum.NetControl.`**RuleType**
Bases: `enum.IntFlag`

Type of rules that the framework supports. Each type lists the extra NetControl::Rule fields it uses, if any.

Plugins may extend this type to define their own.

c.f. base/frameworks/netcontrol/types.zeek

DROP = 1

MODIFY = 2

REDIRECT = 4

WHITELIST = 8

class zlogging.enum.NetControl.**TargetType**
Bases: enum.IntFlag

Type defining the target of a rule.

Rules can either be applied to the forward path, affecting all network traffic, or on the monitor path, only affecting the traffic that is sent to Zeek. The second is mostly used for shunting, which allows Zeek to tell the networking hardware that it wants to no longer see traffic that it identified as benign.

c.f. base/frameworks/netcontrol/types.zeek

FORWARD = 1

MONITOR = 2

class zlogging.enum.NetControl.**CatchReleaseActions**
Bases: enum.IntFlag

The enum that contains the different kinds of messages that are logged by catch and release.

c.f. policy/frameworks/netcontrol/catch-and-release.zeek

ADDED = 2

DROP = 4

DROPPED = 8

FORGOTTEN = 32

INFO = 1

SEEN_AGAIN = 64

UNBLOCK = 16

Notice Namespace

Namespace: Notice.

class zlogging.enum.Notice.**Action**
Bases: enum.IntFlag

These are values representing actions that can be taken with notices.

c.f. base/frameworks/notice/main.zeek

ACTION_ADD_GEODATA = 64

ACTION_ALARM = 8

ACTION_DROP = 128

ACTION_EMAIL = 4

ACTION_EMAIL_ADMIN = 16

ACTION_LOG = 2

ACTION_NONE = 1

ACTION_PAGE = 32

class zlogging.enum.Notice.**Type**
Bases: enum.IntFlag

Scripts creating new notices need to redef this enum to add their own specific notice types which would then get used when they call the NOTICE function. The convention is to give a general category along with the specific notice separating words with underscores and using leading capitals on each word except for abbreviations which are kept in all capitals. For example, SSH::Password_Guessing is for hosts that have crossed a threshold of failed SSH logins.

c.f. base/frameworks/notice/main.zeek

CaptureLoss__Too_Much_Loss = 524288

Conn__Content_Gap = 16777216

Conn__Retransmission_Inconsistency = 8388608

DNS__External_Name = 33554432

FTP__Bruteforcing = 67108864

FTP__Site_Exec_Success = 134217728

HTTP__SQL_Injection_Attacker = 268435456

HTTP__SQL_Injection_Victim = 536870912

Heartbleed__SSL_Heartbeat_Attack = 1099511627776

Heartbleed__SSL_Heartbeat_Attack_Success = 2199023255552

Heartbleed__SSL_Heartbeat_Many_Requests = 8796093022208

Heartbleed__SSL_Heartbeat_Odd_Length = 4398046511104

Intel__Notice = 8192

PacketFilter__Cannot_BPF_Shunt_Conn = 65536

PacketFilter__Compile_Failure = 128

PacketFilter__Dropped_Packets = 1024

PacketFilter__Install_Failure = 256

PacketFilter__No_More_Conn_Shunts_Available = 32768

PacketFilter__Too_Long_To_Compile_Filter = 512

ProtocolDetector__Protocol_Found = 2048

ProtocolDetector__Server_Found = 4096

SMTP__Blocklist_Blocked_Host = 2147483648

SMTP__Blocklist_Error_Message = 1073741824

SMTP__Suspicious_Origination = 4294967296

SSH__Interesting_Hostname_Login = 68719476736

SSH__Login_By_Password_Guesser = 17179869184

SSH__Password_Guessing = 8589934592

SSH__Watched_Country_Login = 34359738368

SSL__Certificate_Expired = 137438953472

SSL__Certificate_Expires_Soon = 274877906944

SSL__Certificate_Not_Valid_Yet = 549755813888

SSL__Invalid_Ocsp_Response = 35184372088832

SSL__Invalid_Server_Cert = 17592186044416

SSL__Old_Version = 140737488355328

SSL__Weak_Cipher = 281474976710656

SSL__Weak_Key = 70368744177664

Scan__Address_Scan = 2097152

Scan__Port_Scan = 4194304

Signatures__Count_Signature = 32

Signatures__Multiple_Sig_Responders = 16

Signatures__Multiple_Signatures = 8

Signatures__Sensitive_Signature = 4

Signatures__Signature_Summary = 64

Software__Software_Version_Change = 131072

Software__Vulnerable_Version = 262144

Tally = 1

TeamCymruMalwareHashRegistry__Match = 16384

Traceroute__Detected = 1048576

Weird__Activity = 2

ZeekygenExample__Zeekygen_Four = 4503599627370496

ZeekygenExample__Zeekygen_One = 562949953421312

ZeekygenExample__Zeekygen_Three = 2251799813685248

ZeekygenExample__Zeekygen_Two = 1125899906842624

`OpenFlow` Namespace

Namespace: `OpenFlow`.

class `zlogging.enum.OpenFlow.`**ofp_action_type**

Bases: `enum.IntFlag`

Openflow action_type definitions.

The openflow action type defines what actions openflow can take to modify a packet

c.f. base/frameworks/openflow/consts.zeek

OFPAT_ENQUEUE = 2048

OFPAT_OUTPUT = 1

OFPAT_SET_DL_DST = 32

OFPAT_SET_DL_SRC = 16

OFPAT_SET_NW_DST = 128

OFPAT_SET_NW_SRC = 64

OFPAT_SET_NW_TOS = 256

OFPAT_SET_TP_DST = 1024

OFPAT_SET_TP_SRC = 512

OFPAT_SET_VLAN_PCP = 4

OFPAT_SET_VLAN_VID = 2

OFPAT_STRIP_VLAN = 8

OFPAT_VENDOR = 4096

class zlogging.enum.OpenFlow.**ofp_config_flags**
Bases: enum.IntFlag

Openflow config flag definitions.

TODO: describe

c.f. base/frameworks/openflow/consts.zeek

OFPC_FRAG_DROP = 2

OFPC_FRAG_MASK = 8

OFPC_FRAG_NORMAL = 1

OFPC_FRAG_REASM = 4

class zlogging.enum.OpenFlow.**ofp_flow_mod_command**
Bases: enum.IntFlag

Openflow flow_mod_command definitions.

The openflow flow_mod_command describes of what kind an action is.

c.f. base/frameworks/openflow/consts.zeek

OFPFC_ADD = 1

OFPFC_DELETE = 8

OFPFC_DELETE_STRICT = 16

OFPFC_MODIFY = 2

OFPFC_MODIFY_STRICT = 4

class zlogging.enum.OpenFlow.**Plugin**
Bases: enum.IntFlag

Available openflow plugins.

c.f. base/frameworks/openflow/types.zeek

BROKER = 8

INVALID = 1

OFLOG = 4

RYU = 2

`ProtocolDetector` Namespace

Namespace: `ProtocolDetector`.

class `zlogging.enum.ProtocolDetector.`**dir**
Bases: `enum.IntFlag`

c.f. policy/frameworks/dpd/detect-protocols.zeek

BOTH = 8

INCOMING = 2

NONE = 1

OUTGOING = 4

`Reporter` Namespace

Namespace: `Reporter`.

class `zlogging.enum.Reporter.`**Level**
Bases: `enum.IntFlag`

c.f. base/bif/types.bif.zeek

ERROR = 4

INFO = 1

WARNING = 2

`SMB` Namespace

Namespace: `SMB`.

class `zlogging.enum.SMB.`**Action**
Bases: `enum.IntFlag`

Abstracted actions for SMB file actions.

c.f. base/protocols/smb/main.zeek

FILE_CLOSE = 8

FILE_DELETE = 16

FILE_OPEN = 4

FILE_READ = 1

FILE_RENAME = 32

FILE_SET_ATTRIBUTE = 64

FILE_WRITE = 2

PIPE_CLOSE = 1024

PIPE_OPEN = 512

PIPE_READ = 128

PIPE_WRITE = 256

PRINT_CLOSE = 16384

PRINT_OPEN = 8192

PRINT_READ = 2048

PRINT_WRITE = 4096

SOCKS Namespace

Namespace: SOCKS.

class zlogging.enum.SOCKS.**RequestType**
Bases: enum.IntFlag

c.f. base/protocols/socks/consts.zeek

CONNECTION = 1

PORT = 2

UDP_ASSOCIATE = 4

SSL Namespace

Namespace: SSL.

class zlogging.enum.SSL.**SctSource**
Bases: enum.IntFlag

List of the different sources for Signed Certificate Timestamp

c.f. policy/protocols/ssl/validate-sct.zeek

SCT_OCSP_EXT = 4

SCT_TLS_EXT = 2

SCT_X509_EXT = 1

Signatures Namespace

Namespace: Signatures.

class zlogging.enum.Signatures.**Action**
Bases: enum.IntFlag

These are the default actions you can apply to signature matches. All of them write the signature record to the logging stream unless declared otherwise.

c.f. base/frameworks/signatures/main.zeek

SIG_ALARM = 16

SIG_ALARM_ONCE = 64

SIG_ALARM_PER_ORIG = 32

SIG_COUNT_PER_RESP = 128

SIG_FILE_BUT_NO_SCAN = 8

SIG_IGNORE = 1

SIG_LOG = 4

SIG_QUIET = 2

SIG_SUMMARY = 256

`Software` Namespace

Namespace: `Software`.

class `zlogging.enum.Software.`**Type**

Bases: `enum.IntFlag`

Scripts detecting new types of software need to redef this enum to add their own specific software types which would then be used when they create Software::Info records.

c.f. base/frameworks/software/main.zeek

DHCP__CLIENT = 8

DHCP__SERVER = 4

FTP__CLIENT = 16

FTP__SERVER = 32

HTTP__APPSERVER = 512

HTTP__BROWSER = 1024

HTTP__BROWSER_PLUGIN = 128

HTTP__SERVER = 256

HTTP__WEB_APPLICATION = 64

MySQL__SERVER = 2048

OS__WINDOWS = 2

SMTP__MAIL_CLIENT = 4096

SMTP__MAIL_SERVER = 8192

SMTP__WEBMAIL_SERVER = 16384

SSH__CLIENT = 65536

SSH__SERVER = 32768

UNKNOWN = 1

`SumStats` Namespace

Namespace: `SumStats`.

class `zlogging.enum.SumStats.`**Calculation**
Bases: `enum.IntFlag`

Type to represent the calculations that are available. The calculations are all defined as plugins.

c.f. base/frameworks/sumstats/main.zeek

AVERAGE = 2

HLL_UNIQUE = 4

LAST = 8

MAX = 16

MIN = 32

PLACEHOLDER = 1

SAMPLE = 64

STD_DEV = 256

SUM = 512

TOPK = 1024

UNIQUE = 2048

VARIANCE = 128

`Tunnel` Namespace

Namespace: `Tunnel`.

class `zlogging.enum.Tunnel.`**Type**
Bases: `enum.IntFlag`

c.f. base/bif/types.bif.zeek

AYIYA = 4

GRE = 128

GTPv1 = 32

HTTP = 64

IP = 2

NONE = 1

SOCKS = 16

TEREDO = 8

VXLAN = 256

class `zlogging.enum.Tunnel.`**Action**
Bases: `enum.IntFlag`

Types of interesting activity that can occur with a tunnel.

c.f. base/frameworks/tunnels/main.zeek

CLOSE = 2

DISCOVER = 1

EXPIRE = 4

Weird Namespace

Namespace: Weird.

class zlogging.enum.Weird.**Action**
Bases: enum.IntFlag

Types of actions that may be taken when handling weird activity events.

c.f. base/frameworks/notice/weird.zeek

ACTION_IGNORE = 2

ACTION_LOG = 4

ACTION_LOG_ONCE = 8

ACTION_LOG_PER_CONN = 16

ACTION_LOG_PER_ORIG = 32

ACTION_NOTICE = 64

ACTION_NOTICE_ONCE = 128

ACTION_NOTICE_PER_CONN = 256

ACTION_NOTICE_PER_ORIG = 512

ACTION_UNSPECIFIED = 1

ZeekygenExample Namespace

Namespace: ZeekygenExample.

class zlogging.enum.ZeekygenExample.**SimpleEnum**
Bases: enum.IntFlag

Documentation for the "SimpleEnum" type goes here. It can span multiple lines.

c.f. zeekygen/example.zeek

FIVE = 16

FOUR = 8

ONE = 1

THREE = 4

TWO = 2

zeek Namespace

Namespace: zeek.

class zlogging.enum.zeek.**layer3_proto**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

L3_ARP = 4

L3_IPV4 = 1

L3_IPV6 = 2

L3_UNKNOWN = 8

class zlogging.enum.zeek.**link_encap**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

LINK_ETHERNET = 1

LINK_UNKNOWN = 2

class zlogging.enum.zeek.**rpc_status**
Bases: enum.IntFlag

c.f. base/bif/types.bif.zeek

RPC_AUTH_ERROR = 256

RPC_GARBAGE_ARGS = 16

RPC_PROC_UNAVAIL = 8

RPC_PROG_MISMATCH = 4

RPC_PROG_UNAVAIL = 2

RPC_SUCCESS = 1

RPC_SYSTEM_ERR = 32

RPC_TIMEOUT = 64

RPC_UNKNOWN_ERROR = 512

RPC_VERS_MISMATCH = 128

class zlogging.enum.zeek.**IPAddrAnonymization**
Bases: enum.IntFlag

See also: anonymize_addr

c.f. base/init-bare.zeek

KEEP_ORIG_ADDR = 1

PREFIX_PRESERVING_A50 = 8

PREFIX_PRESERVING_MD5 = 16

RANDOM_MD5 = 4

SEQUENTIALLY_NUMBERED = 2

class zlogging.enum.zeek.**IPAddrAnonymizationClass**
Bases: enum.IntFlag

See also: anonymize_addr

c.f. base/init-bare.zeek

ORIG_ADDR = 1

OTHER_ADDR = 4

RESP_ADDR = 2

class zlogging.enum.zeek.**PcapFilterID**
Bases: enum.IntFlag

Enum type identifying dynamic BPF filters. These are used by Pcap::precompile_pcap_filter and Pcap::precompile_pcap_filter.

c.f. base/init-bare.zeek

None = 1

PacketFilter__DefaultPcapFilter = 2

PacketFilter__FilterTester = 4

class zlogging.enum.zeek.**pkt_profile_modes**
Bases: enum.IntFlag

Output modes for packet profiling information.

See also: pkt_profile_mode, pkt_profile_freq, pkt_profile_file

c.f. base/init-bare.zeek

PKT_PROFILE_MODE_BYTES = 8

PKT_PROFILE_MODE_NONE = 1

PKT_PROFILE_MODE_PKTS = 4

PKT_PROFILE_MODE_SECS = 2

class zlogging.enum.zeek.**transport_proto**
Bases: enum.IntFlag

A connection's transport-layer protocol. Note that Zeek uses the term "connection" broadly, using flow semantics for ICMP and UDP.

c.f. base/init-bare.zeek

icmp = 8

tcp = 2

udp = 4

unknown_transport = 1

class zlogging.enum.zeek.**Direction**
Bases: enum.IntFlag

c.f. base/utils/directions-and-hosts.zeek

BIDIRECTIONAL = 4

INBOUND = 1

NO_DIRECTION = 8

OUTBOUND = 2

class zlogging.enum.zeek.**Host**
Bases: enum.IntFlag

c.f. base/utils/directions-and-hosts.zeek

ALL_HOSTS = 4

LOCAL_HOSTS = 1

NO_HOSTS = 8

REMOTE_HOSTS = 2

1.2 Module Contents

Bro/Zeek logging framework.

zlogging.**write**(*data*, *filename*, *format*, **args*, ***kwargs*)
Write Bro/Zeek log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **filename** (*os.PathLike*) – Log file name.
- **format** (*str*) – Log format.
- ***args** – See *write_json()* and *write_ascii()* for more information.
- ****kwargs** – See *write_json()* and *write_ascii()* for more information.

Raises ***WriterFormatError*** – If format is not supported.

zlogging.**dump**(*data*, *file*, *format*, **args*, ***kwargs*)
Write Bro/Zeek log file.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **format** (*str*) – Log format.
- **file** (*_io.TextIOWrapper*) – Log file object opened in text mode.
- ***args** – See *dump_json()* and *dump_ascii()* for more information.
- ****kwargs** – See *dump_json()* and *dump_ascii()* for more information.

Raises ***WriterFormatError*** – If format is not supported.

zlogging.**dumps**(*data*, *format*, **args*, ***kwargs*)
Write Bro/Zeek log string.

Parameters

- **data** (Iterable of *Model*) – Log records as an Iterable of *Model* per line.
- **format** (*str*) – Log format.
- ***args** – See *dumps_json()* and *dumps_ascii()* for more information.
- ****kwargs** – See *dumps_json()* and *dumps_ascii()* for more information.

Raises ***WriterFormatError*** – If `format` is not supported.

`zlogging.`**`parse`**(*filename*, **args*, ***kwargs*)
Parse Bro/Zeek log file.

Parameters

- **filename** (*os.PathLike*) – Log file name.
- ***args** – See *parse_json()* and *parse_ascii()* for more information.
- ****kwargs** – See *parse_json()* and *parse_ascii()* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

`zlogging.`**`load`**(*file*, **args*, ***kwargs*)
Parse Bro/Zeek log file.

Parameters

- **file** (*_io.BufferedReader*) – Log file object opened in binary mode.
- ***args** – See *load_json()* and *load_ascii()* for more information.
- ****kwargs** – See *load_json()* and *load_ascii()* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

`zlogging.`**`loads`**(*data*, **args*, ***kwargs*)
Parse Bro/Zeek log string.

Parameters

- **data** (*AnyStr*) – Log string as binary or encoded string.
- ***args** – See *loads_json()* and *loads_ascii()* for more information.
- ****kwargs** – See *loads_json()* and *loads_ascii()* for more information.

Returns The parsed JSON log data.

Raises ***ParserError*** – If the format of the log file is unknown.

Return type Union[*zlogging._data.JSONInfo*, *zlogging._data.ASCIIInfo*]

class `zlogging.`**`Model`**(**args*, ***kwargs*)
Bases: `object`

Log data model.

Variables

- **__fields__** (`OrderedDict` mapping `str` and *BaseType*) – Fields of the data model.
- **__record_fields__** (`OrderedDict` mapping `str` and `RecordType`) – Fields of `record` data type in the data model.
- **__empty_field__** (*bytes*) – Placeholder for empty field.
- **__unset_field__** (*bytes*) – Placeholder for unset field.
- **__set_separator__** (*bytes*) – Separator for set/vector fields.

Warns **BroDeprecationWarning** – Use of `bro_*` type annotations.

Raises

- ***ModelValueError*** – In case of inconsistency between field data types, or values of `unset_field`, `empty_field` and `set_separator`.
- ***ModelTypeError*** – Wrong parameters when initialisation.

Note: Customise the `Model.__post_init__` method in your subclassed data model to implement your own ideas.

Example

Define a custom log data model using the prefines Bro/Zeek data types, or subclasses of *BaseType*:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

Or you may use type annotations as PEP 484 introduced when declaring data models. All available type hints can be found in `typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

However, when mixing annotations and direct assignments, annotations will take proceedings, i.e. the `Model` class shall process first annotations then assignments. Should there be any conflicts, `ModelError` will be raised.

See also:

See `_aux_expand_typing()` for more information about processing the fields.

property fields
`OrderedDict` mapping `str` and *BaseType*: fields of the data model

property unset_field
bytes: placeholder for empty field

property empty_field
bytes: placeholder for unset field

property set_separator
bytes: separator for set/vector fields

__post_init__()
Post-processing customisation.

__call__(*format*)
Serialise data model with given format.

Parameters **format** (*str*) – Serialisation format.

Returns The serialised data.

Raises ***ModelFormatError*** – If `format` is not supproted, i.e. `Mode.to{format}()` does not exist.

Return type Any

tojson()
Serialise data model as JSON log format.

Returns An `OrderedDict` mapping each field and serialised JSON serialisable data.

Return type OrderedDict[str, Any]

toascii()
Serialise data model as ASCII log format.

Returns An `OrderedDict` mapping each field and serialised text data.

Return type OrderedDict[str, str]

asdict(*dict_factory=None*)
Convert data model as a dictionary mapping field names to field values.

Parameters **dict_factory** (*Optional[type]*) – If given, `dict_factory` will be used instead of built-in `dict`.

Returns A dictionary mapping field names to field values.

Return type Dict[str, Any]

astuple(*tuple_factory=None*)
Convert data model as a tuple of field values.

Parameters **tuple_factory** (*Optional[type]*) – If given, `tuple_factory` will be used instead of built-in `tuple`.

Returns A tuple of field values.

Return type Tuple[Any]

zlogging.**new_model**(*name*, ***fields*)
Create a data model dynamically with the appropriate fields.

Parameters

- **name** (*str*) – data model name
- ****fields** – defined fields of the data model
- **Any] fields** (*Dict[str,*) –

Returns created data model

Return type `Model`

Examples

Typically, we define a data model by subclassing the `Model` class, as following:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

when defining dynamically with `new_model()`, the definition above can be rewrote to:

```
MyLog = new_model('MyLog', field_one=StringType(), field_two=SetType(element_
↪type=PortType))
```

---

class zlogging.**AddrType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `addr` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The parsed IP address. If `data` is *unset*, `None` will be returned.

Return type Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The JSON serialisable IP address string.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ipaddress.IPv4Address, ipaddress.IPv6Address]*) – raw data

Returns The ASCII representation of the IP address.

Return type str

class zlogging.**BoolType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `bool` data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for set/vector fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse data from string.

Parameters **data** (*Union[AnyStr, bool]*) – raw data

Returns The parsed boolean data. If data is *unset*, None will be returned.

Raises ***ZeekValueError*** – If data is NOT *unset* and NOT T (True) nor F (False) in Bro/Zeek script language.

Return type Union[None, bool]

tojson(*data*)
Serialize data as JSON log format.

Parameters **data** (*Union[None, bool]*) – raw data

Returns The JSON serialisable boolean data.

Return type Union[None, bool]

toascii(*data*)
Serialize data as ASCII log format.

Parameters **data** (*Union[None, bool]*) – raw data

Returns T if True, F if False.

Return type str

class zlogging.**CountType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek count data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_ulong]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_ulong]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_ulong]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_ulong]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.**DoubleType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `double` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, decimal.Decimal]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, decimal.Decimal]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, decimal.Decimal]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, decimal.Decimal]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.**EnumType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *namespaces=None*, *bare=False*, *enum_hook=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `enum` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- **namespaces** (`List[str]`, optional) – Namespaces to be loaded.
- **bare** (`bool`, optional) – If `True`, do not load `zeek` namespace by default.
- **enum_hook** (`dict` mapping of `str` and `enum.Enum`, optional) – Additional enum to be included in the namespace.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **enum_namespaces** (`dict` mapping `str` and `enum.Enum`) – Global namespace for `enum` data type.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, enum.Enum]*) – raw data

Returns The parsed enum data. If `data` is *unset*, `None` will be returned.

Warns **ZeekValueWarning** – If `date` is not defined in the enum namespace.

Return type Union[None, enum.Enum]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, enum.Enum]*) – raw data

Returns The JSON serialisable enum data.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, enum.Enum]*) – raw data

Returns The ASCII representation of the enum data.

Return type str

class zlogging.**IntervalType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `interval` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, datetime.timedelta]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, datetime.timedelta]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, datetime.timedelta]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, datetime.timedelta]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.**IntType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `int` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_long]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_long]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_long]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_long]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.**PortType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek `port` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, ctypes.c_ushort]*) – raw data

Returns The parsed port number. If `data` is *unset*, `None` will be returned.

Return type Union[None, ctypes.c_ushort]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ctypes.c_ushort]*) – raw data

Returns The JSON serialisable port number string.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ctypes.c_ushort]*) – raw data

Returns The ASCII representation of the port number.

Return type str

class zlogging.**RecordType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***element_mapping*)

Bases: *zlogging.types._VariadicType*

Bro/Zeek `record` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – element_mapping (`dict` mapping `str` and *BaseType* instance): Data type of container's elements.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **element_mapping** (`dict` mapping `str` and *BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If `element_mapping` is not supplied.
- ***ZeekValueError*** – If `element_mapping` is not a valid Bro/Zeek data type; or in case of inconsistency from `empty_field`, `unset_field` and `set_separator` of each field.

Note: A valid `element_mapping` should be a *simple* or *generic* data type, i.e. a subclass of *_SimpleType* or *_GenericType*.

See also:

See `_aux_expand_typing()` for more information about processing the fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

class zlogging.**SetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)

Bases: *zlogging.types._GenericType*, `typing.Generic`

Bro/Zeek `set` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.

- **element_type** (*BaseType* instance) – Data type of container's elements.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If `element_type` is not supplied.
- ***ZeekValueError*** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
>>> SetType[StringType]
```

which is the same **at runtime** as following:

```
>>> SetType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *_SimpleType*.

---

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, Set[data]]*) – raw data

Returns The parsed set data. If `data` is *unset*, `None` will be returned.

Return type Union[None, Set[data]]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, Set[data]]*) – raw data

Returns The JSON serialisable set data.

Return type list

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, Set[data]]*) – raw data

Returns The ASCII representation of the set data.

Return type str

class zlogging.**StringType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `string` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, memoryview, bytearray]*) – raw data

Returns The parsed string data. If `data` is *unset*, `None` will be returned.

Return type Union[None, ByteString]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, ByteString]*) – raw data

Returns The JSON serialisable string data encoded in ASCII.

Return type str

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, ByteString]*) – raw data

Returns The ASCII encoded string data.

Return type str

class zlogging.**SubnetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *zlogging.types._SimpleType*

Bro/Zeek `subnet` data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for set/vector fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse data from string.

Parameters **data** (*Union[AnyStr, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The parsed IP network. If data is *unset*, None will be returned.

Return type Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]

tojson(*data*)
Serialize data as JSON log format.

Parameters **data** (*Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The JSON serialisable IP network string.

Return type str

toascii(*data*)
Serialize data as ASCII log format.

Parameters **data** (*Union[None, ipaddress.IPv4Network, ipaddress.IPv6Network]*) – raw data

Returns The ASCII representation of the IP network.

Return type str

class zlogging.**TimeType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)
Bases: *zlogging.types._SimpleType*

Bro/Zeek time data type.

Parameters

- **empty_field** (bytes or str, optional) – Placeholder for empty field.
- **unset_field** (bytes or str, optional) – Placeholder for unset field.
- **set_separator** (bytes or str, optional) – Separator for set/vector fields.

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse(*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, datetime.datetime]*) – raw data

Returns The parsed numeral data. If `data` is *unset*, `None` will be returned.

Return type Union[None, datetime.datetime]

tojson(*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, datetime.datetime]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, datetime.datetime]*) – raw data

Returns The ASCII representation of numeral data.

Return type str

class zlogging.**VectorType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)
Bases: *zlogging.types._GenericType*, typing.Generic

Bro/Zeek `vector` data type.

Parameters

- **empty_field** (`bytes` or `str`, optional) – Placeholder for empty field.
- **unset_field** (`bytes` or `str`, optional) – Placeholder for unset field.
- **set_separator** (`bytes` or `str`, optional) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.

- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.
- **element_type** (*BaseType* instance) – Data type of container's elements.

Raises

- ***ZeekTypeError*** – If `element_type` is not supplied.
- ***ZeekValueError*** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced PEP 484:

```
>>> VectorType[StringType]
```

which is the same **at runtime** as following:

```
>>> VectorType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *_SimpleType*.

---

property python_type
type: Corresponding Python type annotation.

property zeek_type
str: Corresponding Zeek type name.

parse (*data*)
Parse `data` from string.

Parameters **data** (*Union[AnyStr, List[data]]*) – raw data

Returns The parsed list data. If `data` is *unset*, `None` will be returned.

Return type Union[None, List[data]]

tojson (*data*)
Serialize `data` as JSON log format.

Parameters **data** (*Union[None, List[data]]*) – raw data

Returns The JSON serialisable list data.

Return type list

toascii (*data*)
Serialize `data` as ASCII log format.

Parameters **data** (*Union[None, List[data]]*) – raw data

Returns The ASCII representation of the list data.

Return type str

The `ZLogging` module provides an easy-to-use bridge between the logging framework of the well-known Bro/Zeek Network Security Monitor (IDS).

As of version 3.0, the `Bro` project has been officially renamed to `Zeek`.[1]

[1] https://blog.zeek.org/2018/10/renaming-bro-project_11.html

It was originally developed and derived from the BroAPT project, which is an APT detection framework based on the Bro/Zeek IDS and extended with highly customised and customisable Python wrappers.

CHAPTER

TWO

INSTALLATION

Note: `ZLogging` supports Python all versions above and includes **3.6**

```
pip install zlogging
```

CHAPTER

THREE

USAGE

Currently `ZLogging` supports the two builtin formats as supported by the Bro/Zeek logging framework, i.e. ASCII and JSON.

A typical ASCII log file would be like:

```
#separator \x09
#set_separator      ,
#empty_field        (empty)
#unset_field        -
#path       http
#open       2020-02-09-18-54-09
#fields     ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
↪      trans_depth     method  host    uri     referrer        version user_agent
↪  origin  request_body_len        response_body_len       status_code     status_
↪msg      info_code       info_msg        tags    username        password
↪proxied orig_fuids      orig_filenames  orig_mime_types resp_fuids      resp_
↪filenames  resp_mime_types
#types      time    string  addr    port    addr    port    count   string  string
↪string  string  string  string  string  count   count   count   string  count
↪string  set[enum]       string  string  set[string]     vector[string]
↪vector[string]  vector[string]  vector[string]  vector[string]  vector[string]
1581245648.761106   CSksID3S6Zxplpvmxg      192.168.2.108   56475   151.139.128.14
↪80      1       GET     ocsp.sectigo.com        /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFEML0g5PE3oabJGPJOXafjJNRzPIBBSNjF7EVK2K4Xfpm/
↪mbBeG4AY1h4QIQfdsAWJ+CXcbhDVFyNWosjQ==       -       1.1     com.apple.trustd/2.0
↪ -       0       471     200     OK      -       -       (empty) -       -       -
↪     -       -       -       FPtlyEAhcf8orBPu7       -       application/ocsp-
↪response
1581245651.379048   CuvUnl4HyhQbCs4tXe      192.168.2.108   56483   23.59.247.10
↪80      1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==
↪     -       1.1     com.apple.trustd/2.0    -       0       1398    200     OK
↪-       -       (empty) -       -       -       -       -
↪FRfFoq3hSZkdCNDf9l      -       application/ocsp-response
1581245654.396334   CWo4pd1z97XLB2o0h2      192.168.2.108   56486   23.59.247.122
↪80      1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==
↪     -       1.1     com.apple.trustd/2.0    -       0       1398    200     OK
↪-       -       (empty) -       -       -       -       -
↪FvQehf1pRsGmwDUzJe      -       application/ocsp-response
1581245692.728840   CxFQzh2ePtsnQhFNX3      192.168.2.108   56527   23.59.247.10
↪80      1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==
↪     -       1.1     com.apple.trustd/2.0    -       0       1398    200     OK
↪-       -       (empty) -       -       -       -       -
↪FIeFj8WWNyhA1psGg       -       application/ocsp-response
```

(continues on next page)

(continued from previous page)

```
1581245701.693971   CPZSNk1Y6kDvAN0KZ8      192.168.2.108   56534   23.59.247.122
↪80      1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==
↪     -       1.1     com.apple.trustd/2.0    -       0       1398    200     OK
↪-       -       (empty) -       -       -       -       -
↪F0fGHe4RPuNBhYWNv6      -       application/ocsp-response
1581245707.848088   Cnab6CHFOprdppKi5       192.168.2.108   56542   23.59.247.122
↪80      1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==
↪     -       1.1     com.apple.trustd/2.0    -       0       1398    200     OK
↪-       -       (empty) -       -       -       -       -
↪FgDBep1h7EPHC8qQB6      -       application/ocsp-response
1581245952.784242   CPNd6t3ofePpdNjErl      192.168.2.108   56821   176.31.225.118
↪80      1       GET     tracker.trackerfix.com  /announce?info_hash=y\x82es"\x1dV\
↪xde|m\xbe"\xe5\xef\xbe\x04\xb3\x1fW\xfc&peer_id=-qB4210-0ZOn5Ifyl*WF&port=63108&
↪uploaded=0&downloaded=0&left=3225455594&corrupt=0&key=6B23B036&event=started&
↪numwant=200&compact=1&no_peer_id=1&supportcrypto=1&redundant=0   -       1.1     -
↪     -       0       0       307     Temporary Redirect      -       -
↪(empty) -       -       -       -       -       -       -       -       -
1581245960.123295   CfAkwf2CFI13b24gqf      192.168.2.108   56889   176.31.225.118
↪80      1       GET     tracker.trackerfix.com  /announce?info_hash=!u7\xdad\x94x\
↪xecS\x80\x89\x04\x9c\x13#\x84M\x1b\xcd\x1a&peer_id=-qB4210-i36iloGe*QT9&port=63108&
↪uploaded=0&downloaded=0&left=1637966572&corrupt=0&key=ECE6637E&event=started&
↪numwant=200&compact=1&no_peer_id=1&supportcrypto=1&redundant=0       -       1.1
↪  -       -       0       0       307     Temporary Redirect      -       -
↪(empty) -       -       -       -       -       -       -       -       -
#close      2020-02-09-19-01-40
```

Its corresponding JSON log file would be like:

```
{"ts": 1581245648.761106, "uid": "CSksID3S6ZxplpvmXg", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56475, "id.resp_h": "151.139.128.14", "id.resp_p": 80, "trans_depth":
↪1, "method": "GET", "host": "ocsp.sectigo.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFEML0g5PE3oabJGPJOXafjJNRzPIBBSNjF7EVK2K4Xfpm/
↪mbBeG4AY1h4QIQfdsAWJ+CXcbhDVFyNWosjQ==", "referrer": "-", "version": "1.1", "user_
↪agent": "com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_
↪body_len": 471, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg
↪": "-", "tags": [], "username": "-", "password": "-", "proxied": null, "orig_fuids
↪": null, "orig_filenames": null, "orig_mime_types": null, "resp_fuids": [
↪"FPtlyEAhcf8orBPu7"], "resp_filenames": null, "resp_mime_types": ["application/ocsp-
↪response"]}
{"ts": 1581245651.379048, "uid": "CuvUnl4HyhQbCs4tXe", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56483, "id.resp_h": "23.59.247.10", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==",
↪"referrer": "-", "version": "1.1", "user_agent": "com.apple.trustd/2.0", "origin":
↪"-", "request_body_len": 0, "response_body_len": 1398, "status_code": 200, "status_
↪msg": "OK", "info_code": null, "info_msg": "-", "tags": [], "username": "-",
↪"password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_
↪mime_types": null, "resp_fuids": ["FRfFoq3hSZkdCNDf9l"], "resp_filenames": null,
↪"resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245654.396334, "uid": "CWo4pd1z97XLB2o0h2", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56486, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪ "method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==",
↪"referrer": "-", "version": "1.1", "user_agent": "com.apple.trustd/2.0", "origin":
↪"-", "request_body_len": 0, "response_body_len": 1398, "status_code": 200, "status_
↪msg": "OK", "info_code": null, "info_msg": "-", "tags": [], "username": "-",
↪"password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_
↪mime_types": null, "resp_fuids": ["FvQehf1pRsGmwDUzJe"], "resp_filenames": null,
```

(continues on next page)

(continued from previous page)


```
{"ts": 1581245692.72884, "uid": "CxFQzh2ePtsnQhFNX3", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56527, "id.resp_h": "23.59.247.10", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==",
↪"referrer": "-", "version": "1.1", "user_agent": "com.apple.trustd/2.0", "origin":
↪"-", "request_body_len": 0, "response_body_len": 1398, "status_code": 200, "status_
↪msg": "OK", "info_code": null, "info_msg": "-", "tags": [], "username": "-",
↪"password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_
↪mime_types": null, "resp_fuids": ["FIeFj8WWNyhA1psGg"], "resp_filenames": null,
↪"resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245701.693971, "uid": "CPZSNk1Y6kDvAN0KZ8", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56534, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪ "method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==",
↪"referrer": "-", "version": "1.1", "user_agent": "com.apple.trustd/2.0", "origin":
↪"-", "request_body_len": 0, "response_body_len": 1398, "status_code": 200, "status_
↪msg": "OK", "info_code": null, "info_msg": "-", "tags": [], "username": "-",
↪"password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_
↪mime_types": null, "resp_fuids": ["F0fGHe4RPuNBhYWNv6"], "resp_filenames": null,
↪"resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245707.848088, "uid": "Cnab6CHFOprdppKi5", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56542, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪ "method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/
↪0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==",
↪"referrer": "-", "version": "1.1", "user_agent": "com.apple.trustd/2.0", "origin":
↪"-", "request_body_len": 0, "response_body_len": 1398, "status_code": 200, "status_
↪msg": "OK", "info_code": null, "info_msg": "-", "tags": [], "username": "-",
↪"password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_
↪mime_types": null, "resp_fuids": ["FgDBep1h7EPHC8qQB6"], "resp_filenames": null,
↪"resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245952.784242, "uid": "CPNd6t3ofePpdNjErl", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56821, "id.resp_h": "176.31.225.118", "id.resp_p": 80, "trans_depth":␣
↪1, "method": "GET", "host": "tracker.trackerfix.com", "uri": "/announce?info_hash=y\
↪\x82es\"\\x1dV\\xde|m\\xbe\"\\xe5\\xef\\xbe\\x04\\xb3\\x1fW\\xfc&peer_id=-qB4210-
↪0ZOn5Ifyl*WF&port=63108&uploaded=0&downloaded=0&left=3225455594&corrupt=0&
↪key=6B23B036&event=started&numwant=200&compact=1&no_peer_id=1&supportcrypto=1&
↪redundant=0", "referrer": "-", "version": "1.1", "user_agent": "-", "origin": "-",
↪"request_body_len": 0, "response_body_len": 0, "status_code": 307, "status_msg":
↪"Temporary Redirect", "info_code": null, "info_msg": "-", "tags": [], "username": "-
↪", "password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null,
↪"orig_mime_types": null, "resp_fuids": null, "resp_filenames": null, "resp_mime_
↪types": null}
{"ts": 1581245960.123295, "uid": "CfAkwf2CFI13b24gqf", "id.orig_h": "192.168.2.108",
↪"id.orig_p": 56889, "id.resp_h": "176.31.225.118", "id.resp_p": 80, "trans_depth":␣
↪1, "method": "GET", "host": "tracker.trackerfix.com", "uri": "/announce?info_hash=!
↪u7\\xdad\\x94x\\xecS\\x80\\x89\\x04\\x9c\\x13#\\x84M\\x1b\\xcd\\x1a&peer_id=-qB4210-
↪i36iloGe*QT9&port=63108&uploaded=0&downloaded=0&left=1637966572&corrupt=0&
↪key=ECE6637E&event=started&numwant=200&compact=1&no_peer_id=1&supportcrypto=1&
↪redundant=0", "referrer": "-", "version": "1.1", "user_agent": "-", "origin": "-",
↪"request_body_len": 0, "response_body_len": 0, "status_code": 307, "status_msg":
↪"Temporary Redirect", "info_code": null, "info_msg": "-", "tags": [], "username": "-
↪", "password": "-", "proxied": null, "orig_fuids": null, "orig_filenames": null,
↪"orig_mime_types": null, "resp_fuids": null, "resp_filenames": null, "resp_mime_
↪types": null}
```

3.1 How to Load/Parse a Log File?

To load (parse) a log file generically, i.e. when you don't know what format the log file is, you can simple call the *parse()*, *load()*, or *loads()* functions:

```
# to parse log at filename
>>> parse('path/to/log')
# to load log from a file object
>>> with open('path/to/log', 'rb') as file:
...     load(file)
# to load log from a string
>>> with open('/path/to/log', 'rb') as file:
...     loads(file.read())
```

Note: When calling *load()*, the file object must be opened in binary mode.

When calling *loads()*, if the `data` suplied is an encoded string (`str`), the function will first try to decode it as a bytestring (`bytes`) with `'ascii'` encoding.

If you do know the format, you may call the specified functions for each format, e.g. *parse_ascii()* and *parse_json()*, etc.

See also:

- *parse_ascii()*
- *parse_json()*
- *load_ascii()*
- *load_json()*
- *loads_ascii()*
- *loads_json()*

If you would like to customise your own parser, just subclass *BaseParser* and implement your own ideas.

3.2 How to Dump/Write a Log File?

Before dumping (writing) a log file, you need to create a log **data model** first. Just like in the Bro/Zeek script language, when customise logging, you need to notify the logging framework with a new log stream. Here, in `ZLogging`, we introduced **data model** for the same purpose.

A **data model** is a subclass of *Model* with fields and data types declared. A typical **data model** can be as following:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

where `field_one` is `string` type, i.e. *StringType*; and `field_two` is `set[port]` types, i.e. *SetType* of *PortType*.

Or you may use type annotations as PEP 484 introduced when declaring **data models**. All available type hints can be found in `typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

See also:

See *BaseType* and *Model* for more information about the data types and data model.

After declaration of your **data model**, you can know dump (write) your log file with the corresponding functions.

See also:

- *write_ascii()*
- *write_json()*
- *dump_ascii()*
- *dump_json()*
- *dumps_ascii()*
- *dumps_json()*

If you would like to customise your own writer, just subclass `BaseWriter` and implement your own ideas.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

Z

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P