ZLogging

Release 0.1.1

Jarry Shaw

Jan 24, 2021

CONTENTS

CHAPTER

ONE

BRO/ZEEK LOGGING FRAMEWORK FOR PYTHON

1.1 Table of Contents

1.1.1 Dumpers

Predefined Dumpers

Bro/Zeek log dumper.

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** (*PathLike[str]*) – Log file name.
- **format** (str) – Log format.
- ***args** – See *write_json()* and *write_ascii()* for more information.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *write_json()* and *write_ascii()* for more information.

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

Return type None

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** (*PathLike[str]*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Return type None

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** (*PathLike[str]*) – Log file name.
- **writer** (*JSONWriter*, optional) – Writer class.
- ***args** – Variable length argument list.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Return type None

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *dumps_json()* and *dumps_ascii()* for more information.

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

Return type str

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type str

Returns The JSON log string.

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Return type str

Returns The JSON log string.

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** (*TextFile*) – Log file object opened in text mode.
- ***args** – See *dump_json()* and *dump_ascii()* for more information.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *dump_json()* and *dump_ascii()* for more information.

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

Return type None

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** (*TextFile*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Return type None

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** (*TextFile*) – Log file object opened in text mode.
- **writer** (*JSONWriter*, optional) – Writer class.
- ***args** – Variable length argument list.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Return type None

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
Log file format.

Type str

Return type str

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

Parameters

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

Return type int

Returns The file offset after writing.

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

Args: file: Log file object opened in text mode. data (*Model*): Log record. lineno: Line number of current line.

Returns: The file offset after writing.

Raises: ASCIIWriterError: If failed to serialise data as ASCII.

w

Return type int

Parameters

- **file** (*TextFile*) –
- **data** (Model) –
- **lineno** (*Optional[int]*) –

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

Parameters

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

Return type int

Returns The file offset after writing.

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

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

Return type int

Returns The file offset after writing.

dump_file (*data=None*, *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.

Return type str

Returns The converted log string.

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.

Return type str

Returns The converted log string.

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

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.

Return type str

Returns The converted log string.

dump_tail()
Serialise trailing fields of ASCII log file.

Return type str

Returns The converted log string.

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

JSON log writer.

property format
Log file format.

Type str

Return type Literal[“json”]

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

Parameters

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

Return type int

Returns The file offset after writing.

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

Parameters

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

Return type int

Returns The file offset after writing.

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

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

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

Return type str

Returns The converted log string.

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.

Return type str

Returns The converted log string.

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

Abstract Base Dumpers

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

Basic log writer.

abstract property format
Log file format.

Type str

Return type str

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

Parameters

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

Return type int

Returns The file offset after writing.

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

Parameters

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

Return type int

Returns The file offset after writing.

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

Parameters

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

Return type int

Returns The file offset after writing.

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

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

Return type str

Returns The converted log string.

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.

Return type str

Returns The converted log string.

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

Parameters

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

Return type int

Returns The file offset after writing.

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

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

Return type str

Returns The converted log string.

1.1.2 Loaders

Predefined Loaders

Bro/Zeek log loader.

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

Parameters

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

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

zlogging.loader.**parse_ascii**(*filename*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)
Parse ASCII log file.

Parameters

- **filename** (*PathLike[str]*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *ASCIIInfo*

Returns The parsed ASCII log data.

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

Parameters

- **filename** (*PathLike[str]*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *JSONInfo*

Returns The parsed JSON log data.

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *ASCIIInfo*

Returns The parsed ASCII log data.

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *JSONInfo*

Returns The parsed JSON log data.

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

Parse Bro/Zeek log file.

Parameters

- **file** (*BinaryFile*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

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

Parse ASCII log file.

Parameters

- **file** (*BinaryFile*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *ASCIIInfo*

Returns The parsed ASCII log data.

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

Parameters

- **file** (*BinaryFile*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type *JSONInfo*

Returns The parsed JSON log data.

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
Log file format.

Type str

Return type Literal[“ascii”]

parse_file(*file*, *model=None*)
Parse log file.

Parameters

- **file** (*BinaryFile*) – Log file object opened in binary mode.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log. This parameter is only kept for API compatibility with its base class BaseLoader, and will **NOT** be used at runtime.

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*, *model=None*, *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.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.
- **separator** (*Optional[bytes]*) – Data separator.
- **parser** (List of *BaseType*, required) – Field data type parsers.

Return type *Model*

Returns The parsed log as a plain dict.

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

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
Log file format.

Type str

Return type Literal["json"]

parse_file(*file*, *model=None*)
Parse log file.

Parameters

- **file** (*BinaryFile*) – Log file object opened in binary mode.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.

Returns

The parsed log as a *Model* per line.

Return type *JSONInfo*

parse_line(*line*, *lineno=0*, *model=None*)
Parse log line as one-line record.

Parameters

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

- **model** (*Optional[Type[Model]]*) – Field declrations of current log.

Return type *Model*

Returns The parsed log as a plain *Model*.

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

Abstract Base Loaders

class zlogging.loader.**BaseParser**

Bases: `object`

Basic log parser.

abstract property format

Log file format.

Type str

Return type `str`

parse(*filename*, *model=None*)

Parse log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.

Return type *Info*

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

abstract parse_file(*file*, *model=None*)

Parse log file.

Parameters

- **file** (*BinaryFile*) – Log file object opened in binary mode.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.

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

Return type *Info*

abstract parse_line(*line*, *lineno=0*, *model=None*)

Parse log line as one-line record.

Parameters

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

Return type *Model*

Returns The parsed log as a plain *Model*.

load(*file*)

Parse log file.

Parameters **file** (*BinaryFile*) – 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.

Return type *Model*

Returns The parsed log as a plain *Model*.

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 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 *Model* class shall process first annotations then assignments. Should there be any conflicts, `ModelError` will be raised.

See also:

See *expand_typing()* for more information about processing the fields.

property fields
fields of the data model

Type `OrderedDict` mapping `str` and *BaseType*

Return type OrderedDict[str, *BaseType*]

property unset_field
placeholder for empty field

Type bytes

Return type `bytes`

property empty_field
placeholder for unset field

Type bytes

Return type `bytes`

property set_separator
separator for set/vector fields

Type bytes

Return type `bytes`

__post_init__()
Post-processing customisation.

Return type `None`

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

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

Return type Any

Returns The serialised data.

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

tojson()
Serialise data model as JSON log format.

Return type OrderedDict[str, Any]

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

toascii()
Serialise data model as ASCII log format.

Return type OrderedDict[str, str]

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

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

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

Return type Dict[str, Any]

Returns A dictionary mapping field names to field values.

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

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

Return type Tuple[Any, ..]

Returns A tuple of field values.

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
- **fields** (*Any*) –

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.**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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type `str`

parse(*data*)

Parse `data` from string.

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

Return type Optional[IPAddress]

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[IPAddress]*) – raw data

Returns The JSON serialisable IP address string.

Return type str

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[IPAddress]*) – raw data

Returns The ASCII representation of the IP address.

Return type str

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

Corresponding Python type annotation.

Type Any

Return type Type[bool]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“bool”]

parse(*data*)

Parse `data` from string.

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

Return type Optional[bool]

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.

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[bool]*) – raw data

Return type Optional[bool]

Returns The JSON serialisable boolean data.

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[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

Corresponding Python type annotation.

Type Any

Return type Type[uint64]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“count”]

parse(*data*)

Parse data from string.

Parameters **data** (*Union[AnyStr, uint64]*) – raw data

Return type Optional[uint64]

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

tojson(*data*)

Serialize data as JSON log format.

Parameters **data** (*Optional[uint64]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)

Serialize data as ASCII log format.

Parameters **data** (*Optional[uint64]*) – 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

Corresponding Python type annotation.

Type Any

Return type Type[Decimal]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal["double"]

parse(*data*)

Parse data from string.

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

Return type Optional[Decimal]

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

tojson(*data*)

Serialize data as JSON log format.

Parameters **data** (*Optional[Decimal]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

toascii(*data*)

Serialize data as ASCII log format.

Parameters **data** (*Optional[Decimal]*) – raw data

Returns The ASCII representation of numeral data.

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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type `str`

parse(*data*)

Parse `data` from string.

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

Return type Optional[Enum]

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

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[Enum]*) – raw data

Returns The JSON serialisable enum data.

Return type str

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

Parameters **data** (*Optional[Enum]*) – raw data

Returns The ASCII representation of the enum 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
Corresponding Python type annotation.

Type Any

Return type Type[TimeDeltaType]

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“interval”]

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

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

Return type Optional[TimeDeltaType]

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

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

Parameters **data** (*Optional[TimeDeltaType]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

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

Parameters **data** (*Optional[TimeDeltaType]*) – 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
Corresponding Python type annotation.

Type Any

Return type Type[int64]

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“int”]

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

Parameters **data** (*Union[AnyStr, int64]*) – raw data

Return type Optional[int64]

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

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

Parameters **data** (*Optional[int64]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

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

Parameters **data** (*Optional[int64]*) – raw data

Returns The ASCII representation of numeral data.

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

Corresponding Python type annotation.

Type Any

Return type Type[uint16]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“port”]

parse(*data*)

Parse `data` from string.

Parameters **data** (*Union[AnyStr, uint16]*) – raw data

Return type Optional[uint16]

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[uint16]*) – raw data

Returns The JSON serialisable port number string.

Return type int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[uint16]*) – raw data

Returns The ASCII representation of the port number.

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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal["record"]

element_mapping: OrderedDict[str, Union[*_SimpleType*, *_GenericType*]]

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

Bases: *zlogging.types._GenericType*, `Generic[zlogging.types._S]`

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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type `str`

parse(*data*)

Parse `data` from string.

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

Return type Optional[Set[_S]]

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

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

Parameters **data** (*Optional[Set[_S]]*) – raw data

Returns The JSON serialisable set data.

Return type list

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

Parameters **data** (*Optional[Set[_S]]*) – raw data

Returns The ASCII representation of the set 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
Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“string”]

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

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

Return type Optional[bytes]

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

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

Parameters **data** (*Optional[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** (*Optional[ByteString]*) – raw data

Returns The ASCII encoded string data.

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
Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type
Corresponding Zeek type name.

Type str

Return type `str`

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

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

Return type Optional[IPNetwork]

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

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

Parameters **data** (*Optional[IPNetwork]*) – raw data

Returns The JSON serialisable IP network string.

Return type str

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

Parameters **data** (*Optional[IPNetwork]*) – raw data

Returns The ASCII representation of the IP network.

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
Corresponding Python type annotation.

Type Any

Return type Type[DateTimeType]

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“time”]

parse(*data*)
Parse data from string.

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

Return type Optional[DateTimeType]

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

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

Parameters **data** (*Optional[DateTimeType]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

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

Parameters **data** (*Optional[DateTimeType]*) – raw data

Returns The ASCII representation of numeral 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*, Generic[zlogging.types._S]

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
Corresponding Python type annotation.

Type Any

Return type Any

property zeek_type
Corresponding Zeek type name.

Type str

Return type str

parse(*data*)
Parse data from string.

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

Return type Optional[List[_S]]

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

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

Parameters **data** (*Optional[List[_S]]*) – raw data

Returns The JSON serialisable list data.

Return type list

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

Parameters **data** (*Optional[List[_S]]*) – raw data

Returns The ASCII representation of the list data.

Return type str

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.

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.

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.

element_mapping: OrderedDict[str, Union[*_SimpleType*, *_GenericType*]]

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

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
Corresponding Python type annotation.

Type Any

Return type Any

abstract property zeek_type
Corresponding Zeek type name.

Type str

Return type str

property bro_type
Corresponding Bro type name.

Type str

Return type str

__call__(*data*)
Parse data from string.

This is a proxy method which calls to *parse()* of the type implementation.

Return type Any

Parameters **data** (*Any*) –

__str__()
Returns the corresponding Zeek type name.

Return type str

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

Return type Any

Parameters **data** (*Any*) –

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

Return type Any

Parameters **data** (*Any*) –

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

Return type str

Parameters **data** (*Any*) –

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.

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.

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.

element_mapping: OrderedDict[str, Union[_SimpleType, _GenericType]]

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

1.1.5 Typing Annotations

Zeek Data Types

zlogging.typing.**zeek_addr**
Zeek `addr` data type.

alias of TypeVar('zeek_addr')

zlogging.typing.**zeek_bool**
Zeek `bool` data type.

alias of TypeVar('zeek_bool')

zlogging.typing.**zeek_count**
Zeek `count` data type.

alias of TypeVar('zeek_count')

zlogging.typing.**zeek_double**
Zeek `double` data type.

alias of TypeVar('zeek_double')

zlogging.typing.**zeek_enum**
Zeek `enum` data type.

alias of TypeVar('zeek_enum')

zlogging.typing.**zeek_interval**
Zeek `interval` data type.

alias of TypeVar('zeek_interval')

zlogging.typing.**zeek_int**
Zeek `int` data type.

alias of TypeVar('zeek_int')

zlogging.typing.**zeek_port**
Zeek `port` data type.

alias of TypeVar('zeek_port')

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**
Zeek `string` data type.

alias of TypeVar('zeek_string')

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

alias of TypeVar('zeek_subnet')

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

alias of TypeVar('zeek_time')

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` data type.

alias of TypeVar('bro_addr')

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

alias of TypeVar('bro_bool')

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

alias of TypeVar('bro_count')

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

alias of TypeVar('bro_double')

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

alias of TypeVar('bro_enum')

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

alias of TypeVar('bro_interval')

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

alias of TypeVar('bro_int')

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

alias of TypeVar('bro_port')

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 data type.

alias of TypeVar('bro_string')

zlogging.typing.**bro_subnet**
Bro subnet data type.

alias of TypeVar('bro_subnet')

zlogging.typing.**bro_time**
Bro time data type.

alias of TypeVar('bro_time')

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.

property format
Log file format.

Type str

Return type Literal["ascii"]

path: PathLike[str]
Log path.

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

Type os.PathLike

open: DateTimeType
Log open time.

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

Type datetime.datetime

close: DateTimeType
Log close time.

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

Type datetime.datetime

data: List[Model]
Log records.

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

Type `list` of *Model*

exit_with_error: bool
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.

property format
Log file format.

Type str

Return type Literal["json"]

data: List[Model]
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
Log file format.

Type str

Return type str

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.

Return type None

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.

Return type None

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.

Return type None

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.

Return type None

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.

Return type None

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.

Return type None

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

Return type None

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** (*BinaryFile*) – 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.

Return type Union[List[str], List[bytes]]

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

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

Parameters

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

Return type str

Returns The converted ASCII string.

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** (*Optional[str]*) – The ASCII representation of infinite numbers (NaN and infinity).

Return type str

Returns The converted ASCII string.

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.

Return type str

Returns The escaped bytestring as an encoded string

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[Type[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**(*value*)
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

NONE = 1

BOOL = 2

INT = 4

COUNT = 8

DOUBLE = 16

STRING = 32

ADDR = 64

SUBNET = 128

PORT = 256

TIME = 512

INTERVAL = 1024

ENUM = 2048

SET = 4096

TABLE = 8192

VECTOR = 16384

class zlogging.enum.Broker.**Type**(*value*)
Bases: enum.IntFlag

The type of a Broker activity being logged.

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

STATUS = 1

ERROR = 2

class zlogging.enum.Broker.**ErrorCode**(*value*)
Bases: enum.IntFlag

Enumerates the possible error types.

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

NO_ERROR = 1

UNSPECIFIED = 2

PEER_INCOMPATIBLE = 4

PEER_INVALID = 8

PEER_UNAVAILABLE = 16

PEER_DISCONNECT_DURING_HANDSHAKE = 32

PEER_TIMEOUT = 64

MASTER_EXISTS = 128

NO_SUCH_MASTER = 256

NO_SUCH_KEY = 512

REQUEST_TIMEOUT = 1024

TYPE_CLASH = 2048

INVALID_DATA = 4096

BACKEND_FAILURE = 8192

STALE_DATA = 16384

CANNOT_OPEN_FILE = 32768

CANNOT_WRITE_FILE = 65536

INVALID_TOPIC_KEY = 131072

END_OF_FILE = 262144

INVALID_TAG = 524288

INVALID_STATUS = 1048576

CAF_ERROR = 2097152

class zlogging.enum.Broker.**PeerStatus**(*value*)

Bases: enum.IntFlag

The possible states of a peer endpoint.

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

INITIALIZING = 1

CONNECTING = 2

CONNECTED = 4

PEERED = 8

DISCONNECTED = 16

RECONNECTING = 32

class zlogging.enum.Broker.**BackendType**(*value*)

Bases: enum.IntFlag

Enumerates the possible storage backends.

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

MEMORY = 1

SQLITE = 2

ROCKSDB = 4

class zlogging.enum.Broker.**QueryStatus**(*value*)
Bases: enum.IntFlag

Whether a data store query could be completed or not.

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

SUCCESS = 1

FAILURE = 2

Cluster Namespace

Namespace: Cluster.

class zlogging.enum.Cluster.**NodeType**(*value*)
Bases: enum.IntFlag

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

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

NONE = 1

CONTROL = 2

LOGGER = 4

MANAGER = 8

PROXY = 16

WORKER = 32

TIME_MACHINE = 64

DCE_RPC Namespace

Namespace: DCE_RPC.

class zlogging.enum.DCE_RPC.**IfID**(*value*)
Bases: enum.IntFlag

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

unknown_if = 1

epmapper = 2

lsarpc = 4

lsa_ds = 8

mgmt = 16

netlogon = 32

samr = 64

srvsvc = 128

spoolss = 256

drs = 512

winspipe = 1024

wkssvc = 2048

oxid = 4096

ISCMActivator = 8192

class zlogging.enum.DCE_RPC.**PType**(*value*)

Bases: enum.IntFlag

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

REQUEST = 1

PING = 2

RESPONSE = 4

FAULT = 8

WORKING = 16

NOCALL = 32

REJECT = 64

ACK = 128

CL_CANCEL = 256

FACK = 512

CANCEL_ACK = 1024

BIND = 2048

BIND_ACK = 4096

BIND_NAK = 8192

ALTER_CONTEXT = 16384

ALTER_CONTEXT_RESP = 32768

AUTH3 = 65536

SHUTDOWN = 131072

CO_CANCEL = 262144

ORPHANED = 524288

RTS = 1048576

`HTTP` Namespace

Namespace: `HTTP`.

class zlogging.enum.HTTP.**Tags**(*value*)
Bases: enum.IntFlag

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

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

EMPTY = 1

URI_SQLI = 2

POST_SQLI = 4

COOKIE_SQLI = 8

`Input` Namespace

Namespace: `Input`.

class zlogging.enum.Input.**Event**(*value*)
Bases: enum.IntFlag

Type that describes what kind of change occurred.

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

EVENT_NEW = 1

EVENT_CHANGED = 2

EVENT_REMOVED = 4

class zlogging.enum.Input.**Mode**(*value*)
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**(*value*)
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**(*value*)
Bases: enum.IntFlag

Enum type to represent various types of intelligence data.

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

ADDR = 1

SUBNET = 2

URL = 4

SOFTWARE = 8

EMAIL = 16

DOMAIN = 32

USER_NAME = 64

CERT_HASH = 128

PUBKEY_HASH = 256

FILE_HASH = 512

FILE_NAME = 1024

class zlogging.enum.Intel.**Where**(*value*)
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

IN_ANYWHERE = 1

Conn__IN_ORIG = 2

Conn__IN_RESP = 4

Files__IN_HASH = 8

Files__IN_NAME = 16

DNS__IN_REQUEST = 32

DNS__IN_RESPONSE = 64

HTTP__IN_HOST_HEADER = 128

HTTP__IN_REFERRER_HEADER = 256

HTTP__IN_USER_AGENT_HEADER = 512

HTTP__IN_X_FORWARDED_FOR_HEADER = 1024

HTTP__IN_URL = 2048

SMTP__IN_MAIL_FROM = 4096

SMTP__IN_RCPT_TO = 8192

SMTP__IN_FROM = 16384

SMTP__IN_TO = 32768

SMTP__IN_CC = 65536

SMTP__IN_RECEIVED_HEADER = 131072

SMTP__IN_REPLY_TO = 262144

SMTP__IN_X_ORIGINATING_IP_HEADER = 524288

SMTP__IN_MESSAGE = 1048576

SSH__IN_SERVER_HOST_KEY = 2097152

SSL__IN_SERVER_NAME = 4194304

SMTP__IN_HEADER = 8388608

X509__IN_CERT = 16777216

SMB__IN_FILE_NAME = 33554432

SSH__SUCCESSFUL_LOGIN = 67108864

`JSON` Namespace

Namespace: `JSON`.

class `zlogging.enum.JSON.`**TimestampFormat**(*value*)

Bases: `enum.IntFlag`

c.f. base/init-bare.zeek

TS_EPOCH = 1

TS_MILLIS = 2

TS_ISO8601 = 4

`Known` Namespace

Namespace: `Known`.

class `zlogging.enum.Known.`**ModbusDeviceType**(*value*)

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**(*value*)
Bases: `enum.IntFlag`

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

AUTO_BPF = 1

`Log` Namespace

Namespace: `Log`.

class `zlogging.enum.Log.`**ID**(*value*)
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

UNKNOWN = 1

PRINTLOG = 2

Broker__LOG = 4

Files__LOG = 8

Reporter__LOG = 16

Cluster__LOG = 32

Notice__LOG = 64

Notice__ALARM_LOG = 128

Weird__LOG = 256

DPD__LOG = 512

Signatures__LOG = 1024

PacketFilter__LOG = 2048

Software__LOG = 4096

Intel__LOG = 8192

Config__LOG = 16384

Tunnel__LOG = 32768

OpenFlow__LOG = 65536

NetControl__LOG = 131072

NetControl__DROP = 262144

NetControl__SHUNT = 524288

Conn__LOG = 1048576

DCE_RPC__LOG = 2097152

```
DHCP__LOG = 4194304
DNP3__LOG = 8388608
DNS__LOG = 16777216
FTP__LOG = 33554432
SSL__LOG = 67108864
X509__LOG = 134217728
HTTP__LOG = 268435456
IRC__LOG = 536870912
KRB__LOG = 1073741824
Modbus__LOG = 2147483648
mysql__LOG = 4294967296
NTLM__LOG = 8589934592
NTP__LOG = 17179869184
RADIUS__LOG = 34359738368
RDP__LOG = 68719476736
RFB__LOG = 137438953472
SIP__LOG = 274877906944
SNMP__LOG = 549755813888
SMB__AUTH_LOG = 1099511627776
SMB__MAPPING_LOG = 2199023255552
SMB__FILES_LOG = 4398046511104
SMTP__LOG = 8796093022208
SOCKS__LOG = 17592186044416
SSH__LOG = 35184372088832
Syslog__LOG = 70368744177664
PE__LOG = 140737488355328
NetControl__CATCH_RELEASE = 281474976710656
Unified2__LOG = 562949953421312
OCSP__LOG = 1125899906842624
Barnyard2__LOG = 2251799813685248
CaptureLoss__LOG = 4503599627370496
Traceroute__LOG = 9007199254740992
LoadedScripts__LOG = 18014398509481984
Stats__LOG = 36028797018963968
WeirdStats__LOG = 72057594037927936
Known__HOSTS_LOG = 144115188075855872
```

Known__SERVICES_LOG = 288230376151711744

Known__MODBUS_LOG = 576460752303423488

Modbus__REGISTER_CHANGE_LOG = 1152921504606846976

MQTT__CONNECT_LOG = 2305843009213693952

MQTT__SUBSCRIBE_LOG = 4611686018427387904

MQTT__PUBLISH_LOG = 9223372036854775808

SMB__CMD_LOG = 18446744073709551616

Known__CERTS_LOG = 36893488147419103232

ZeekygenExample__LOG = 73786976294838206464

class zlogging.enum.Log.**PrintLogType**(*value*)
Bases: enum.IntFlag

Configurations for Log::print_to_log

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

REDIRECT_NONE = 1

REDIRECT_STDOUT = 2

REDIRECT_ALL = 4

class zlogging.enum.Log.**Writer**(*value*)
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**(*value*)
Bases: enum.IntFlag

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

AUTH_NULL = 1

AUTH_UNIX = 2

AUTH_SHORT = 4

AUTH_DES = 8

class zlogging.enum.MOUNT3.**proc_t**(*value*)
Bases: enum.IntFlag

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

PROC_NULL = 1

PROC_MNT = 2

PROC_DUMP = 4

PROC_UMNT = 8

PROC_UMNT_ALL = 16

PROC_EXPORT = 32

PROC_END_OF_PROCS = 64

class zlogging.enum.MOUNT3.**status_t**(*value*)
Bases: enum.IntFlag

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

MNT3_OK = 1

MNT3ERR_PERM = 2

MNT3ERR_NOENT = 4

MNT3ERR_IO = 8

MNT3ERR_ACCES = 16

MNT3ERR_NOTDIR = 32

MNT3ERR_INVAL = 64

MNT3ERR_NAMETOOLONG = 128

MNT3ERR_NOTSUPP = 256

MNT3ERR_SERVERFAULT = 512

MOUNT3ERR_UNKNOWN = 1024

MQTT Namespace

Namespace: MQTT.

class zlogging.enum.MQTT.**SubUnsub**(*value*)
Bases: enum.IntFlag

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

SUBSCRIBE = 1

UNSUBSCRIBE = 2

NFS3 Namespace

Namespace: NFS3.

class zlogging.enum.NFS3.**createmode_t**(*value*)
Bases: enum.IntFlag

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

UNCHECKED = 1

GUARDED = 2

EXCLUSIVE = 4

class zlogging.enum.NFS3.**file_type_t**(*value*)

Bases: enum.IntFlag

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

FTYPE_REG = 1

FTYPE_DIR = 2

FTYPE_BLK = 4

FTYPE_CHR = 8

FTYPE_LNK = 16

FTYPE_SOCK = 32

FTYPE_FIFO = 64

class zlogging.enum.NFS3.**proc_t**(*value*)

Bases: enum.IntFlag

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

PROC_NULL = 1

PROC_GETATTR = 2

PROC_SETATTR = 4

PROC_LOOKUP = 8

PROC_ACCESS = 16

PROC_READLINK = 32

PROC_READ = 64

PROC_WRITE = 128

PROC_CREATE = 256

PROC_MKDIR = 512

PROC_SYMLINK = 1024

PROC_MKNOD = 2048

PROC_REMOVE = 4096

PROC_RMDIR = 8192

PROC_RENAME = 16384

PROC_LINK = 32768

PROC_READDIR = 65536

PROC_READDIRPLUS = 131072

PROC_FSSTAT = 262144

PROC_FSINFO = 524288

PROC_PATHCONF = 1048576

PROC_COMMIT = 2097152

PROC_END_OF_PROCS = 4194304

class zlogging.enum.NFS3.**stable_how_t**(*value*)
Bases: enum.IntFlag

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

UNSTABLE = 1

DATA_SYNC = 2

FILE_SYNC = 4

class zlogging.enum.NFS3.**status_t**(*value*)
Bases: enum.IntFlag

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

NFS3ERR_OK = 1

NFS3ERR_PERM = 2

NFS3ERR_NOENT = 4

NFS3ERR_IO = 8

NFS3ERR_NXIO = 16

NFS3ERR_ACCES = 32

NFS3ERR_EXIST = 64

NFS3ERR_XDEV = 128

NFS3ERR_NODEV = 256

NFS3ERR_NOTDIR = 512

NFS3ERR_ISDIR = 1024

NFS3ERR_INVAL = 2048

NFS3ERR_FBIG = 4096

NFS3ERR_NOSPC = 8192

NFS3ERR_ROFS = 16384

NFS3ERR_MLINK = 32768

NFS3ERR_NAMETOOLONG = 65536

NFS3ERR_NOTEMPTY = 131072

NFS3ERR_DQUOT = 262144

NFS3ERR_STALE = 524288

NFS3ERR_REMOTE = 1048576

NFS3ERR_BADHANDLE = 2097152

NFS3ERR_NOT_SYNC = 4194304

NFS3ERR_BAD_COOKIE = 8388608

NFS3ERR_NOTSUPP = 16777216

NFS3ERR_TOOSMALL = 33554432

NFS3ERR_SERVERFAULT = 67108864

NFS3ERR_BADTYPE = 134217728

NFS3ERR_JUKEBOX = 268435456

NFS3ERR_UNKNOWN = 536870912

class zlogging.enum.NFS3.**time_how_t**(*value*)
Bases: enum.IntFlag

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

DONT_CHANGE = 1

SET_TO_SERVER_TIME = 2

SET_TO_CLIENT_TIME = 4

NetControl Namespace

Namespace: NetControl.

class zlogging.enum.NetControl.**InfoCategory**(*value*)
Bases: enum.IntFlag

Type of an entry in the NetControl log.

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

MESSAGE = 1

ERROR = 2

RULE = 4

class zlogging.enum.NetControl.**InfoState**(*value*)
Bases: enum.IntFlag

State of an entry in the NetControl log.

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

REQUESTED = 1

SUCCEEDED = 2

EXISTS = 4

FAILED = 8

REMOVED = 16

TIMEOUT = 32

class zlogging.enum.NetControl.**EntityType**(*value*)
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**(*value*)
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**(*value*)
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**(*value*)
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

INFO = 1

ADDED = 2

DROP = 4

DROPPED = 8

UNBLOCK = 16

FORGOTTEN = 32

SEEN_AGAIN = 64

Notice Namespace

Namespace: Notice.

class zlogging.enum.Notice.**Action**(*value*)
Bases: enum.IntFlag

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

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

ACTION_NONE = 1

ACTION_LOG = 2

ACTION_EMAIL = 4

ACTION_ALARM = 8

ACTION_EMAIL_ADMIN = 16

ACTION_PAGE = 32

ACTION_ADD_GEODATA = 64

ACTION_DROP = 128

class zlogging.enum.Notice.**Type**(*value*)
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

Tally = 1

Weird__Activity = 2

Signatures__Sensitive_Signature = 4

Signatures__Multiple_Signatures = 8

Signatures__Multiple_Sig_Responders = 16

Signatures__Count_Signature = 32

Signatures__Signature_Summary = 64

PacketFilter__Compile_Failure = 128

PacketFilter__Install_Failure = 256

PacketFilter__Too_Long_To_Compile_Filter = 512

PacketFilter__Dropped_Packets = 1024

ProtocolDetector__Protocol_Found = 2048

ProtocolDetector__Server_Found = 4096

Intel__Notice = 8192

TeamCymruMalwareHashRegistry__Match = 16384

PacketFilter__No_More_Conn_Shunts_Available = 32768

PacketFilter__Cannot_BPF_Shunt_Conn = 65536

Software__Software_Version_Change = 131072

Software__Vulnerable_Version = 262144

CaptureLoss__Too_Much_Loss = 524288

Traceroute__Detected = 1048576

Scan__Address_Scan = 2097152

Scan__Port_Scan = 4194304

Conn__Retransmission_Inconsistency = 8388608

```
Conn__Content_Gap = 16777216
DNS__External_Name = 33554432
FTP__Bruteforcing = 67108864
FTP__Site_Exec_Success = 134217728
HTTP__SQL_Injection_Attacker = 268435456
HTTP__SQL_Injection_Victim = 536870912
SMTP__Blocklist_Error_Message = 1073741824
SMTP__Blocklist_Blocked_Host = 2147483648
SMTP__Suspicious_Origination = 4294967296
SSH__Password_Guessing = 8589934592
SSH__Login_By_Password_Guesser = 17179869184
SSH__Watched_Country_Login = 34359738368
SSH__Interesting_Hostname_Login = 68719476736
SSL__Certificate_Expired = 137438953472
SSL__Certificate_Expires_Soon = 274877906944
SSL__Certificate_Not_Valid_Yet = 549755813888
Heartbleed__SSL_Heartbeat_Attack = 1099511627776
Heartbleed__SSL_Heartbeat_Attack_Success = 2199023255552
Heartbleed__SSL_Heartbeat_Odd_Length = 4398046511104
Heartbleed__SSL_Heartbeat_Many_Requests = 8796093022208
SSL__Invalid_Server_Cert = 17592186044416
SSL__Invalid_Ocsp_Response = 35184372088832
SSL__Weak_Key = 70368744177664
SSL__Old_Version = 140737488355328
SSL__Weak_Cipher = 281474976710656
ZeekygenExample__Zeekygen_One = 562949953421312
ZeekygenExample__Zeekygen_Two = 1125899906842624
ZeekygenExample__Zeekygen_Three = 2251799813685248
ZeekygenExample__Zeekygen_Four = 4503599627370496
```

`OpenFlow` Namespace

Namespace: `OpenFlow`.

class zlogging.enum.OpenFlow.**ofp_action_type**(*value*)
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_OUTPUT = 1

OFPAT_SET_VLAN_VID = 2

OFPAT_SET_VLAN_PCP = 4

OFPAT_STRIP_VLAN = 8

OFPAT_SET_DL_SRC = 16

OFPAT_SET_DL_DST = 32

OFPAT_SET_NW_SRC = 64

OFPAT_SET_NW_DST = 128

OFPAT_SET_NW_TOS = 256

OFPAT_SET_TP_SRC = 512

OFPAT_SET_TP_DST = 1024

OFPAT_ENQUEUE = 2048

OFPAT_VENDOR = 4096

class zlogging.enum.OpenFlow.**ofp_config_flags**(*value*)
Bases: enum.IntFlag

Openflow config flag definitions.

TODO: describe

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

OFPC_FRAG_NORMAL = 1

OFPC_FRAG_DROP = 2

OFPC_FRAG_REASM = 4

OFPC_FRAG_MASK = 8

class zlogging.enum.OpenFlow.**ofp_flow_mod_command**(*value*)
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_MODIFY = 2

OFPFC_MODIFY_STRICT = 4

OFPFC_DELETE = 8

OFPFC_DELETE_STRICT = 16

class zlogging.enum.OpenFlow.**Plugin**(*value*)
Bases: enum.IntFlag

Available openflow plugins.

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

INVALID = 1

RYU = 2

OFLOG = 4

BROKER = 8

ProtocolDetector Namespace

Namespace: ProtocolDetector.

class zlogging.enum.ProtocolDetector.**dir**(*value*)
Bases: enum.IntFlag

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

NONE = 1

INCOMING = 2

OUTGOING = 4

BOTH = 8

Reporter Namespace

Namespace: Reporter.

class zlogging.enum.Reporter.**Level**(*value*)
Bases: enum.IntFlag

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

INFO = 1

WARNING = 2

ERROR = 4

SMB Namespace

Namespace: SMB.

class zlogging.enum.SMB.**Action**(*value*)
Bases: enum.IntFlag

Abstracted actions for SMB file actions.

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

FILE_READ = 1

FILE_WRITE = 2

FILE_OPEN = 4

FILE_CLOSE = 8

FILE_DELETE = 16

FILE_RENAME = 32

FILE_SET_ATTRIBUTE = 64

PIPE_READ = 128

PIPE_WRITE = 256

PIPE_OPEN = 512

PIPE_CLOSE = 1024

PRINT_READ = 2048

PRINT_WRITE = 4096

PRINT_OPEN = 8192

PRINT_CLOSE = 16384

SOCKS Namespace

Namespace: SOCKS.

class zlogging.enum.SOCKS.**RequestType**(*value*)

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**(*value*)

Bases: enum.IntFlag

List of the different sources for Signed Certificate Timestamp

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

SCT_X509_EXT = 1

SCT_TLS_EXT = 2

SCT_OCSP_EXT = 4

`Signatures` Namespace

Namespace: `Signatures`.

class `zlogging.enum.Signatures.`**Action**(*value*)
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_IGNORE = 1

SIG_QUIET = 2

SIG_LOG = 4

SIG_FILE_BUT_NO_SCAN = 8

SIG_ALARM = 16

SIG_ALARM_PER_ORIG = 32

SIG_ALARM_ONCE = 64

SIG_COUNT_PER_RESP = 128

SIG_SUMMARY = 256

`Software` Namespace

Namespace: `Software`.

class `zlogging.enum.Software.`**Type**(*value*)
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

UNKNOWN = 1

OS__WINDOWS = 2

DHCP__SERVER = 4

DHCP__CLIENT = 8

FTP__CLIENT = 16

FTP__SERVER = 32

HTTP__WEB_APPLICATION = 64

HTTP__BROWSER_PLUGIN = 128

HTTP__SERVER = 256

HTTP__APPSERVER = 512

HTTP__BROWSER = 1024

MySQL__SERVER = 2048

SMTP__MAIL_CLIENT = 4096

SMTP__MAIL_SERVER = 8192

SMTP__WEBMAIL_SERVER = 16384

SSH__SERVER = 32768

SSH__CLIENT = 65536

`SumStats` Namespace

Namespace: `SumStats`.

class zlogging.enum.SumStats.**Calculation**(*value*)
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

PLACEHOLDER = 1

AVERAGE = 2

HLL_UNIQUE = 4

LAST = 8

MAX = 16

MIN = 32

SAMPLE = 64

VARIANCE = 128

STD_DEV = 256

SUM = 512

TOPK = 1024

UNIQUE = 2048

`Tunnel` Namespace

Namespace: `Tunnel`.

class zlogging.enum.Tunnel.**Type**(*value*)
Bases: `enum.IntFlag`

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

NONE = 1

IP = 2

AYIYA = 4

TEREDO = 8

SOCKS = 16

GTPv1 = 32

HTTP = 64

GRE = 128

VXLAN = 256

class zlogging.enum.Tunnel.**Action**(*value*)
Bases: enum.IntFlag

Types of interesting activity that can occur with a tunnel.

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

DISCOVER = 1

CLOSE = 2

EXPIRE = 4

Weird Namespace

Namespace: Weird.

class zlogging.enum.Weird.**Action**(*value*)
Bases: enum.IntFlag

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

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

ACTION_UNSPECIFIED = 1

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

ZeekygenExample Namespace

Namespace: ZeekygenExample.

class zlogging.enum.ZeekygenExample.**SimpleEnum**(*value*)
Bases: enum.IntFlag

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

c.f. zeekygen/example.zeek

ONE = 1

TWO = 2

THREE = 4

FOUR = 8

FIVE = 16

zeek Namespace

Namespace: zeek.

class zlogging.enum.zeek.**TableChange**(*value*)
Bases: enum.IntFlag

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

TABLE_ELEMENT_NEW = 1

TABLE_ELEMENT_CHANGED = 2

TABLE_ELEMENT_REMOVED = 4

TABLE_ELEMENT_EXPIRED = 8

class zlogging.enum.zeek.**layer3_proto**(*value*)
Bases: enum.IntFlag

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

L3_IPV4 = 1

L3_IPV6 = 2

L3_ARP = 4

L3_UNKNOWN = 8

class zlogging.enum.zeek.**link_encap**(*value*)
Bases: enum.IntFlag

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

LINK_ETHERNET = 1

LINK_UNKNOWN = 2

class zlogging.enum.zeek.**rpc_status**(*value*)
Bases: enum.IntFlag

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

RPC_SUCCESS = 1

RPC_PROG_UNAVAIL = 2

RPC_PROG_MISMATCH = 4

RPC_PROC_UNAVAIL = 8

RPC_GARBAGE_ARGS = 16

RPC_SYSTEM_ERR = 32

RPC_TIMEOUT = 64

RPC_VERS_MISMATCH = 128

RPC_AUTH_ERROR = 256

RPC_UNKNOWN_ERROR = 512

class zlogging.enum.zeek.**IPAddrAnonymization**(*value*)
Bases: enum.IntFlag

See also: anonymize_addr

c.f. base/init-bare.zeek

KEEP_ORIG_ADDR = 1

SEQUENTIALLY_NUMBERED = 2

RANDOM_MD5 = 4

PREFIX_PRESERVING_A50 = 8

PREFIX_PRESERVING_MD5 = 16

class zlogging.enum.zeek.**IPAddrAnonymizationClass**(*value*)
Bases: enum.IntFlag

See also: anonymize_addr

c.f. base/init-bare.zeek

ORIG_ADDR = 1

RESP_ADDR = 2

OTHER_ADDR = 4

class zlogging.enum.zeek.**PcapFilterID**(*value*)
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**(*value*)
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_NONE = 1

PKT_PROFILE_MODE_SECS = 2

PKT_PROFILE_MODE_PKTS = 4

PKT_PROFILE_MODE_BYTES = 8

class zlogging.enum.zeek.**transport_proto**(*value*)
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

unknown_transport = 1

tcp = 2

udp = 4

icmp = 8

class zlogging.enum.zeek.**Direction**(*value*)
Bases: enum.IntFlag

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

INBOUND = 1

OUTBOUND = 2

BIDIRECTIONAL = 4

NO_DIRECTION = 8

class zlogging.enum.zeek.**Host**(*value*)
Bases: enum.IntFlag

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

LOCAL_HOSTS = 1

REMOTE_HOSTS = 2

ALL_HOSTS = 4

NO_HOSTS = 8

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** (*PathLike[str]*) – Log file name.
- **format** (str) – Log format.
- ***args** – See *write_json()* and *write_ascii()* for more information.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *write_json()* and *write_ascii()* for more information.

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

Return type None

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** (*TextFile*) – Log file object opened in text mode.
- ***args** – See *dump_json()* and *dump_ascii()* for more information.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *dump_json()* and *dump_ascii()* for more information.

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

Return type None

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Keyword Arguments ****kwargs** – See *dumps_json()* and *dumps_ascii()* for more information.

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

Return type str

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

Parameters

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

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

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

Parameters

- **file** (*BinaryFile*) – 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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

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.
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type Union[*JSONInfo*, *ASCIIInfo*]

Returns The parsed JSON log data.

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

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 `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 `Model` class shall process first annotations then assignments. Should there be any conflicts, `ModelError` will be raised.

See also:

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

property fields
fields of the data model

Type `OrderedDict` mapping `str` and *`BaseType`*

Return type OrderedDict[str, *BaseType*]

property unset_field
placeholder for empty field

Type bytes

Return type `bytes`

property empty_field
placeholder for unset field

Type bytes

Return type `bytes`

property set_separator
separator for set/vector fields

Type bytes

Return type `bytes`

__post_init__()
Post-processing customisation.

Return type `None`

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

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

Return type Any

Returns The serialised data.

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

tojson()
Serialise data model as JSON log format.

Return type OrderedDict[str, Any]

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

toascii()
Serialise data model as ASCII log format.

Return type OrderedDict[str, str]

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

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

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

Return type Dict[str, Any]

Returns A dictionary mapping field names to field values.

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

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

Return type Tuple[Any, ..]

Returns A tuple of field values.

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
- **fields** (*Any*) –

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

Corresponding Python type annotation.

Type Any

Return type Any

property zeek_type

Corresponding Zeek type name.

Type str

Return type str

parse(*data*)

Parse data from string.

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

Return type Optional[IPAddress]

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

tojson(*data*)

Serialize data as JSON log format.

Parameters **data** (*Optional[IPAddress]*) – raw data

Returns The JSON serialisable IP address string.

Return type str

toascii(*data*)

Serialize data as ASCII log format.

Parameters **data** (*Optional[IPAddress]*) – 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

Corresponding Python type annotation.

Type Any

Return type Type[bool]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“bool”]

parse(*data*)

Parse `data` from string.

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

Return type Optional[bool]

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.

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[bool]*) – raw data

Return type Optional[bool]

Returns The JSON serialisable boolean data.

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[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

Corresponding Python type annotation.

Type Any

Return type Type[uint64]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“count”]

parse(*data*)

Parse `data` from string.

Parameters **data** (*Union[AnyStr, uint64]*) – raw data

Return type Optional[uint64]

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[uint64]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[uint64]*) – 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

Corresponding Python type annotation.

Type Any

Return type Type[Decimal]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal["double"]

parse(*data*)

Parse data from string.

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

Return type Optional[Decimal]

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

tojson(*data*)

Serialize data as JSON log format.

Parameters **data** (*Optional[Decimal]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

toascii(*data*)

Serialize data as ASCII log format.

Parameters **data** (*Optional[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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type `str`

parse(*data*)

Parse `data` from string.

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

Return type Optional[Enum]

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

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[Enum]*) – raw data

Returns The JSON serialisable enum data.

Return type str

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

Parameters **data** (*Optional[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
Corresponding Python type annotation.

Type Any

Return type Type[TimeDeltaType]

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“interval”]

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

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

Return type Optional[TimeDeltaType]

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

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

Parameters **data** (*Optional[TimeDeltaType]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

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

Parameters **data** (*Optional[TimeDeltaType]*) – 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

Corresponding Python type annotation.

Type Any

Return type Type[int64]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“int”]

parse(*data*)

Parse `data` from string.

Parameters **data** (*Union[AnyStr, int64]*) – raw data

Return type Optional[int64]

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[int64]*) – raw data

Returns The JSON serialisable numeral data.

Return type int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[int64]*) – 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

Corresponding Python type annotation.

Type Any

Return type Type[uint16]

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal[“port”]

parse(*data*)

Parse `data` from string.

Parameters **data** (*Union[AnyStr, uint16]*) – raw data

Return type Optional[uint16]

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

tojson(*data*)

Serialize `data` as JSON log format.

Parameters **data** (*Optional[uint16]*) – raw data

Returns The JSON serialisable port number string.

Return type int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters **data** (*Optional[uint16]*) – 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

Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type

Corresponding Zeek type name.

Type str

Return type Literal["record"]

element_mapping: OrderedDict[str, Union[_SimpleType, _GenericType]]

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

Bases: *zlogging.types._GenericType*, `Generic[zlogging.types._S]`

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
Corresponding Python type annotation.

Type Any

Return type Any

property zeek_type
Corresponding Zeek type name.

Type str

Return type str

parse(*data*)
Parse data from string.

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

Return type Optional[Set[_S]]

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

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

Parameters **data** (*Optional[Set[_S]]*) – raw data

Returns The JSON serialisable set data.

Return type list

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

Parameters **data** (*Optional[Set[_S]]*) – 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
Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“string”]

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

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

Return type Optional[bytes]

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

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

Parameters **data** (*Optional[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** (*Optional[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
Corresponding Python type annotation.

Type Any

Return type `Any`

property zeek_type
Corresponding Zeek type name.

Type str

Return type `str`

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

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

Return type Optional[IPNetwork]

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

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

Parameters **data** (*Optional[IPNetwork]*) – raw data

Returns The JSON serialisable IP network string.

Return type str

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

Parameters **data** (*Optional[IPNetwork]*) – 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
Corresponding Python type annotation.

Type Any

Return type Type[DateTimeType]

property zeek_type
Corresponding Zeek type name.

Type str

Return type Literal[“time”]

parse(*data*)
Parse data from string.

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

Return type Optional[DateTimeType]

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

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

Parameters **data** (*Optional[DateTimeType]*) – raw data

Returns The JSON serialisable numeral data.

Return type float

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

Parameters **data** (*Optional[DateTimeType]*) – 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*, Generic[zlogging.types._S]

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
Corresponding Python type annotation.

Type Any

Return type Any

property zeek_type
Corresponding Zeek type name.

Type str

Return type str

parse(*data*)
Parse data from string.

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

Return type Optional[List[_S]]

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

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

Parameters **data** (*Optional[List[_S]]*) – raw data

Returns The JSON serialisable list data.

Return type list

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

Parameters **data** (*Optional[List[_S]]*) – 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]

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.

[1] https://blog.zeek.org/2018/10/renaming-bro-project_11.html

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 `zlogging.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

R

S

T

U

V

W

Z