pyosmocom-usermanual

Harald Welte

Sep 23, 2024

CONTENTS:

Within the Osmocom (Open Source Mobile Communciations) project family, there have been a number of implementations of key protocols or interfaces in the C language, and occasionally also Erlang.

This repository contains python implementation of key Osmocom related library code, like

- utilities for common problems found in mobile communications
- TLV parsers/encoders
- helpers for 'construct' based encoders/decoders

CHAPTER

ONE

TABLE OF CONTENTS

1.1 osmocom.utils

various pyosmocom utilities osmocom: various utilities

class osmocom.utils.**JsonEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Extend the standard library JSONEncoder with support for more types.

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

osmocom.utils.**all_subclasses**(*cls*) → set

Recursively get all subclasses of a specified class

osmocom.utils.**auto_int**(*x*)

Helper function for argparse to accept hexadecimal integers.

osmocom.utils.**b2h**(*b: bytearray*) → Hexstr

convert from a sequence of bytes to a string of hex nibbles

osmocom.utils.**h2b**(*s: Hexstr*) → bytearray

convert from a string of hex nibbles to a sequence of bytes

osmocom.utils.**h2i**(*s: Hexstr*) → List[int]

convert from a string of hex nibbles to a list of integers

osmocom.utils.**h2s**(*s: Hexstr*) → str

convert from a string of hex nibbles to an ASCII string

osmocom.utils.**i2h**(*s: List[int]*) → Hexstr

convert from a list of integers to a string of hex nibbles

osmocom.utils.**i2s**(*s: List[int]*) → str

convert from a list of integers to an ASCII string

osmocom.utils.**is_decimal**(*instr: str*) → str

Method that can be used as 'type' in argparse.add_argument() to validate the value consists of an even sequence of decimal digits only.

osmocom.utils.**is_hex**(*string: str*, *minlen: int = 2*, *maxlen: int | None = None*) → bool

Check if a string is a valid hexstring

osmocom.utils.**is_hexstr**(*instr: str*) → str

Method that can be used as 'type' in argparse.add_argument() to validate the value consists of an even sequence of hexadecimal digits only.

osmocom.utils.**is_hexstr_or_decimal**(*instr: str*) → str

Method that can be used as 'type' in argparse.add_argument() to validate the value consists of [hexa]decimal digits only.

osmocom.utils.**lpad**(*s: str*, *l: int*, *c='f'*) → str

pad string on the left side. :param s: string to pad :param l: total length to pad to :param c: padding character

Returns

String 's' padded with as many 'c' as needed to reach total length of 'l'

osmocom.utils.**rpad**(*s: str*, *l: int*, *c='f'*) → str

pad string on the right side. :param s: string to pad :param l: total length to pad to :param c: padding character

Returns
String 's' padded with as many 'c' as needed to reach total length of 'l'

osmocom.utils.**s2h**(*s: str*) → Hexstr

convert from an ASCII string to a string of hex nibbles

osmocom.utils.**str_sanitize**(*s: str*) → str

replace all non printable chars, line breaks and whitespaces, with ' ', make sure that there are no whitespaces at the end and at the beginning of the string.

Parameters
s – string to sanitize

Returns
filtered result of string 's'

osmocom.utils.**swap_nibbles**(*s: Hexstr*) → Hexstr

swap the nibbles in a hex string

1.2 osmocom.construct

Helpers for the *consturct* declarative parser/encoder Utility code related to the integration of the 'construct' declarative parser.

class osmocom.construct.**BcdAdapter**(*subcon*)

convert a bytes() type to a string of BCD nibbles.

osmocom.construct.**BitsRFU**(*n=1*)

Field that packs Reserved for Future Use (RFU) bit(s) as defined in TS 31.101 Sec. "3.4 Coding Conventions"

Use this for (currently) unused/reserved bits whose contents should be initialized automatically but should not be cleared in the future or when restoring read data (unlike padding).

Parameters
n (*Integer*) – Number of bits (default: 1)

osmocom.construct.**BytesRFU**(*n=1*)

Field that packs Reserved for Future Use (RFU) byte(s) as defined in TS 31.101 Sec. "3.4 Coding Conventions"

Use this for (currently) unused/reserved bytes whose contents should be initialized automatically but should not be cleared in the future or when restoring read data (unlike padding).

Parameters
n (*Integer*) – Number of bytes (default: 1)

class osmocom.construct.**DnsAdapter**(*subcon*)

Convert between DNS label format (length-prefixed labels) and string format.

class osmocom.construct.**GreedyInteger**(*signed=False*, *swapped=False*, *minlen=0*)

A variable-length integer implementation, think of combining GrredyBytes with BytesInteger.

class osmocom.construct.**GsmOrUcs2Adapter**(*subcon*)

Try to encode into a GSM 03.38 string; if that fails, fall back to UCS-2 as described in TS 102 221 Annex A.

osmocom.construct.**GsmOrUcs2String**(*n*)

GSM 03.38 or UCS-2 (TS 102 221 Annex A) encoded byte string of fixed length n. Encoder appends padding bytes (b'xff') to maintain length. Decoder removes those trailing bytes.

Exceptions are raised for invalid characters and length excess.

Parameters
n (*Integer*) – Fixed length of the encoded byte string

osmocom.construct.**GsmString**(*n*)

GSM 03.38 encoded byte string of fixed length n. Encoder appends padding bytes (b'xff') to maintain length. Decoder removes those trailing bytes.

Exceptions are raised for invalid characters and length excess.

Parameters
n (*Integer*) – Fixed length of the encoded byte string

class osmocom.construct.**GsmStringAdapter**(*subcon*, *codec='gsm03.38'*, *err='strict'*)

Convert GSM 03.38 encoded bytes to a string.

class osmocom.construct.**HexAdapter**(*subcon*)

convert a bytes() type to a string of hex nibbles.

class osmocom.construct.**InvertAdapter**(*subcon*)

inverse logic (false->true, true->false).

class osmocom.construct.**Ipv4Adapter**(*subcon*)

Encoder converts from 4 bytes to string representation (A.B.C.D). Decoder converts from string representation (A.B.C.D) to four bytes.

class osmocom.construct.**Ipv6Adapter**(*subcon*)

Encoder converts from 16 bytes to string representation. Decoder converts from string representation to 16 bytes.

class osmocom.construct.**MultiplyAdapter**(*subcon*, *multiplicator*)

Decoder multiplies by multiplicator Encoder divides by multiplicator

Parameters

- **subcon** – Subconstruct as defined by construct library
- **multiplier** – Multiplier to apply to raw encoded value

class osmocom.construct.**PaddedBcdAdapter**(*subcon*)

Representatin of a BCD string of potentially odd number of BCD digits, which then need to be padded at the end with an 'f' nibble.

class osmocom.construct.**PlmnAdapter**(*subcon*)

convert a bytes(3) type to BCD string like 262-02 or 262-002.

class osmocom.construct.**Rpad**(*subcon*, *pattern=b'\xff'*, *num_per_byte=1*)

Encoder appends padding bytes (b'xff') or characters up to target size. Decoder removes trailing padding bytes/characters.

Parameters

- **subcon** – Subconstruct as defined by construct library
- **pattern** – set padding pattern (default: b'xff')
- **num_per_byte** – number of 'elements' per byte. E.g. for hex nibbles: 2

class osmocom.construct.**StripTrailerAdapter**(*subcon*, *total_length: int*, *default_value=b'\x00'*, *min_len=1*)

Encoder removes all trailing bytes matching the default_value Decoder pads input data up to total_length with default_value

This is used in constellations like "FlagsEnum(StripTrailerAdapter(GreedyBytes, 3), ..." where you have a bitmask that may have 1, 2 or 3 bytes, depending on whether or not any of the LSBs are actually set.

class osmocom.construct.**Ucs2Adapter**(*subcon*)

convert a bytes() type that contains UCS2 encoded characters encoded as defined in TS 102 221 Annex A to normal python string representation (and back).

class osmocom.construct.**Utf8Adapter**(*subcon*)

convert a bytes() type that contains utf8 encoded text to human readable text.

osmocom.construct.**build_construct**(*c*, *decoded_data*, *context: dict = {}*)

Helper function to handle total_len.

osmocom.construct.**filter_dict**(*d*, *exclude_prefix='_'*)

filter the input dict to ensure no keys starting with 'exclude_prefix' remain.

osmocom.construct.**normalize_construct**(*c*, *exclude_prefix: str = '_'*)

Convert a construct specific type to a related base type, mostly useful so we can serialize it.

osmocom.construct.**parse_construct**(*c*, *raw_bin_data: bytes*, *length: int | None = None*, *exclude_prefix: str = '_'*, *context: dict = {}*)

Helper function to wrap around normalize_construct() and filter_dict().

1.3 osmocom.tlv

TLV parser/encoder for BER-TLV, DGI and COMPREHENSION-TLV TLV parser/encoder library supporting various formats.

class osmocom.tlv.**BER_TLV_IE**(***kwargs*)

TLV_IE formatted as ASN.1 BER described in ITU-T X.690 8.1.2.

class osmocom.tlv.**COMPACT_TLV_IE**(***kwargs*)

TLV_IE formatted as COMPACT-TLV described in ISO 7816

to_tlv(*context: dict = {}*)

Convert the internal representation to binary TLV bytes.

class osmocom.tlv.**COMPR_TLV_IE**(***kwargs*)

TLV_IE formated as COMPREHENSION-TLV as described in ETSI TS 101 220.

is_tag_compatible(*rawtag: int*) → bool

Override is_tag_compatible as we need to mask out the comprehension bit when doing compares.

class osmocom.tlv.**ComprTlvMeta**(*name*, *bases*, *namespace*, ***kwargs*)

class osmocom.tlv.**DGI_TLV_IE**(***kwargs*)

TLV_IE formated as GlobalPlatform Systems Scripting Language Specification v1.1.0 Annex B.

class osmocom.tlv.**IE**(***kwargs*)

Base class for various Information Elements. We understand the notion of a hierarchy of IEs on top of the Transcodable class.

child_by_name(*name: str*) → *IE* | None

Return a child IE instance of given snake-case/json type name. This only works in case there is no more than one child IE of the given type.

child_by_type(*cls*) → *IE* | None

Return a child IE instance of given type (class). This only works in case there is no more than one child IE of the given type.

from_bytes(*do: bytes*, *context: dict = {}*)

Parse *the value part* from binary bytes to internal representation.

from_dict(*decoded: dict*)

Set the IE internal decoded representation to data from the argument. If this is a nested IE, the child IE instance list is re-created.

This method is symmetrical to to_dict() above, i.e. the outer dict must contain just a single key-value pair, where the key is the snake-reformatted type name of 'self'

from_val_dict(*decoded*)

Set the IE internal decoded representation to data from the argument. If this is a nested IE, the child IE instance list is re-created.

This method is symmetrical to to_val_dict() aboe, i.e. there is no outer dict containig the snake-reformatted type name of 'self'.

is_constructed()

Is this IE constructed by further nested IEs?

to_bytes(*context: dict = {}*) → bytes

Convert the internal representation *of the value part* to binary bytes.

to_dict()

Return a JSON-serializable dict representing the [nested] IE data. The returned data contains an outer dict with the snake-reformatted type of 'self' and is hence sufficient to re-create an object from it.

abstract to_ie(*context: dict = {}*) → bytes

Convert the internal representation to entire IE including IE header.

to_val_dict()

Return a JSON-serializable dict representing just the [nested] value portion of the IE data. This does not include any indication of the type of 'self', so the resulting dict alone will be insufficient ot recreate an object from it without additional type information.

class osmocom.tlv.**TLV_IE**(***kwargs*)

Abstract base class for various TLV type Information Elements.

is_tag_compatible(*rawtag*) → bool

Is the given rawtag compatible with this class?

to_ie(*context: dict = {}*)

Convert the internal representation to entire IE including IE header.

to_tlv(*context: dict = {}*)

Convert the internal representation to binary TLV bytes.

class osmocom.tlv.**TLV_IE_Collection**(*desc=None*, ***kwargs*)

A TLV_IE_Collection consists of multiple TLV_IE classes identified by their tags. A given encoded DO may contain any of them in any order, and may contain multiple instances of each DO.

from_bytes(*binary: bytes*, *context: dict = {}*) → List[*TLV_IE*]

Create a list of TLV_IEs from the collection based on binary input data. :param binary: binary bytes of encoded data

Returns
list of instances of TLV_IE sub-classes containing parsed data

from_dict(*decoded: List[dict]*) → List[*TLV_IE*]

Create a list of TLV_IE instances from the collection based on an array of dicts, where they key indicates the name of the TLV_IE subclass to use.

class osmocom.tlv.**TlvCollectionMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Metaclass which we use to set some class variables at the time of defining a subclass. This allows us to create subclasses for each Collection type, where the class represents fixed parameters like the nested IE classes and instances of it represent the actual TLV data.

class osmocom.tlv.**TlvMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Metaclass which we use to set some class variables at the time of defining a subclass. This allows us to create subclasses for each TLV/IE type, where the class represents fixed parameters like the tag/type and instances of it represent the actual TLV data.

class osmocom.tlv.**Transcodable**

from_bytes(*do: bytes*, *context: dict = {}*)

Convert from binary bytes to internal representation. Store the decoded result in the internal state and return it.

to_bytes(*context: dict = {}*) → bytes

Convert from internal representation to binary bytes. Store the binary result in the internal state and return it.

osmocom.tlv.**bertlv_encode_len**(*length: int*) → bytes

Encode a single Length value according to ITU-T X.690 8.1.3; only the definite form is supported here. :param length: length value to be encoded

Returns
binary output data of BER-TLV length field

osmocom.tlv.**bertlv_encode_tag**(*t*) → bytes

Encode a single Tag value according to ITU-T X.690 8.1.2

osmocom.tlv.**bertlv_parse_len**(*binary: bytes*) → Tuple[int, bytes]

Parse a single Length value according to ITU-T X.690 8.1.3; only the definite form is supported here. :param binary: binary input data of BER-TLV length field

Returns
Tuple of (length, remainder)

osmocom.tlv.**bertlv_parse_one**(*binary: bytes*) → Tuple[dict, int, bytes, bytes]

Parse a single TLV IE at the start of the given binary data. :param binary: binary input data of BER-TLV length field

Returns
dict, len:int, remainder:bytes)

Return type
Tuple of (tag

osmocom.tlv.**bertlv_parse_one_rawtag**(*binary: bytes*) → Tuple[int, int, bytes, bytes]

Parse a single TLV IE at the start of the given binary data; return tag as raw integer. :param binary: binary input data of BER-TLV length field

Returns
int, len:int, remainder:bytes)

Return type
Tuple of (tag

osmocom.tlv.**bertlv_parse_tag**(*binary: bytes*) → Tuple[dict, bytes]

Parse a single Tag value according to ITU-T X.690 8.1.2 :param binary: binary input data of BER-TLV length field

Returns
int, constructed:bool, tag:int}, remainder:bytes)

Return type
Tuple of ({class

osmocom.tlv.**bertlv_parse_tag_raw**(*binary: bytes*) → Tuple[int, bytes]

Get a single raw Tag from start of input according to ITU-T X.690 8.1.2 :param binary: binary input data of BER-TLV length field

Returns: Tuple of (tag:int, remainder:bytes)

osmocom.tlv.**bertlv_return_one_rawtlv**(*binary: bytes*) → Tuple[int, int, bytes, bytes]

Return one single [encoded] TLV IE at the start of the given binary data. :param binary: binary input data of BER-TLV length field

Returns
int, len:int, tlv:bytes, remainder:bytes)

Return type
Tuple of (tag

osmocom.tlv.**comprehensiontlv_encode_tag**(*tag*) → bytes

Encode a single Tag according to ETSI TS 101 220 Section 7.1.1

osmocom.tlv.**comprehensiontlv_parse_one**(*binary: bytes*) → Tuple[dict, int, bytes, bytes]

Parse a single TLV IE at the start of the given binary data. :param binary: binary input data of BER-TLV length field

Returns
dict, len:int, remainder:bytes)

Return type
Tuple of (tag

osmocom.tlv.**comprehensiontlv_parse_tag**(*binary: bytes*) → Tuple[dict, bytes]

Parse a single Tag according to ETSI TS 101 220 Section 7.1.1

osmocom.tlv.**comprehensiontlv_parse_tag_raw**(*binary: bytes*) → Tuple[int, bytes]

Parse a single Tag according to ETSI TS 101 220 Section 7.1.1

osmocom.tlv.**dgi_encode_len**(*length: int*) → bytes

Encode a single Length value according to GlobalPlatform Systems Scripting Language Specification v1.1.0 Annex B. :param length: length value to be encoded

Returns
binary output data of encoded length field

osmocom.tlv.**dgi_parse_len**(*binary: bytes*) → Tuple[int, bytes]

Parse a single Length value according to GlobalPlatform Systems Scripting Language Specification v1.1.0 Annex B. :param binary: binary input data of BER-TLV length field

Returns
Tuple of (length, remainder)

osmocom.tlv.**flatten_dict_lists**(*inp*)

hierarchically flatten each list-of-dicts into a single dict. This is useful to make the output of hierarchical TLV decoder structures flatter and more easy to read.

1.4 osmocom.gsmtap

GSMTAP pseudo-header for protocol traces Osmocom GSMTAP python implementation. GSMTAP is a packet format used for conveying a number of different telecom-related protocol traces over UDP.

class osmocom.gsmtap.**GsmtapMessage**(*encoded=None*)

Class whose objects represent a single GSMTAP message. Can encode and decode messages.

class osmocom.gsmtap.**GsmtapReceiver**(*bind_ip: str = '127.0.0.1'*, *bind_port: int = 4729*)

Simple receive-side socket implementation for GSMTAP messages.

read_packet() → *GsmtapMessage*

Perform a blocking read on the GSMTAP socket and decode it as GSMTAP message.

1.5 osmocom.gsup

GSUP protocol (Osmocom TCAP/MAP alternative) This is an encoder/decoder implementation for the Osmocom GSUP protocol, built upon the osmocom.tlv and osmocom.construct infrastructure.

class osmocom.gsup.message.**GSUP_TLV_IE**(***kwargs*)

Class representing the TLV format as used in Osmocom GSUP. It's a simple '8-bit tag / 8-bit length / value' variant.

class osmocom.gsup.message.**GsupMessage**(*msg_type:* MsgType | *int*)

Represents a single message within the GSUP protocol.

classmethod from_bytes(*encoded: bytes*) → *GsupMessage*

Create a GsupMessage instance from the decode of the given bytes.

classmethod from_dict(*decoded: dict*) → *GsupMessage*

Create a GsupMessage instance from the decoded dict.

to_bytes()

Encode a GsupMessage instance into bytes.

to_dict()

Encode a GsupMessage instance into a json-serializable dict.

class osmocom.gsup.message.**MsgType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

GSUP protocol message type. Keep this in sync with https://gitea.osmocom.org/osmocom/libosmocore/src/branch/master/include/osmocom/gsm/gsup.h

class osmocom.gsup.message.**tlv**

TLV definitions for the GSUP IEs, utilizing the osmocom.tlv code. Keep this in sync with https://gitea.osmocom.org/osmocom/libosmocore/src/branch/master/include/osmocom/gsm/gsup.h

class **AnApdu**(***kwargs*)

class **AuthTuple**(***kwargs*)

nested_collection_cls

alias of auto_collection_AuthTuple

class **CancelType**(***kwargs*)

class **Cause**(***kwargs*)

class **CauseBssap**(***kwargs*)

class **CauseRr**(***kwargs*)

class **CauseSm**(***kwargs*)

class **CnDomain**(***kwargs*)

class **CurrentRatType**(***kwargs*)

class **DestinationName**(***kwargs*)

class **FreezePTMSI**(***kwargs*)

class **HlrNumber**(***kwargs*)

class **IMEI**(***kwargs*)

class **IMSI**(***kwargs*)

class **IeCollection**(*desc=None*, ***kwargs*)

class **ImeiCheckResult**(***kwargs*)

class **MSISDN**(***kwargs*)

class **MessageClass**(***kwargs*)

class **NumVectorsReq**(***kwargs*)

class **PCO**(***kwargs*)

class **PdpInfo**(***kwargs*)

nested_collection_cls

alias of auto_collection_PdpInfo

class **PdpInfoCompl**(***kwargs*)

class **SessionId**(***kwargs*)

class **SessionState**(***kwargs*)

class **SmAlert**(***kwargs*)

class SmRpCause(**kwargs)

class SmRpDa(**kwargs)

class SmRpMms(**kwargs)

class SmRpMr(**kwargs)

class SmRpOa(**kwargs)

class SmRpUi(**kwargs)

class SourceName(**kwargs)

class SupplementaryServiceInfo(**kwargs)

class SupportedRatTypes(**kwargs)

class auth

Nested TLV IEs within the AuthTuple.

class AUTN(**kwargs)

class AUTS(**kwargs)

class CK(**kwargs)

class IK(**kwargs)

class Kc(**kwargs)

class RAND(**kwargs)

class RES(**kwargs)

class SRES(**kwargs)

class pdp

Nested TLV IEs within the PdpInfo.

class AccessPointName(**kwargs)

class PdpAddress(**kwargs)

class PdpChargingCharacteristics(**kwargs)

class PdpContextId(**kwargs)

class Qos(**kwargs)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

U