astm Documentation

Release 0.5.1-dev

Alexander Shorin

September 09, 2015

Contents

Contents:

CHAPTER 1

`astm.constants` :: ASTM constant values

CHAPTER 2

astm.codec :: Base decoding and encoding functions

astm.codec.**decode**(*data*, *encoding='latin-1'*)

Common ASTM decoding function that tries to guess which kind of data it handles.

If *data* starts with STX character (0x02) than probably it is full ASTM message with checksum and other system characters.

If *data* starts with digit character (0-9) than probably it is frame of records leading by his sequence number. No checksum is expected in this case.

Otherwise it counts *data* as regular record structure.

Note, that *data* should be bytes, not unicode string even if you know his *encoding*.

Parameters

- **data** (*bytes*) – ASTM data object.
- **encoding** (*str*) – Data encoding.

Returns List of ASTM records with unicode data.

Return type list

astm.codec.**decode_component**(*field*, *encoding*)

Decodes ASTM field component.

astm.codec.**decode_frame**(*frame*, *encoding*)

Decodes ASTM frame: list of records followed by sequence number.

astm.codec.**decode_message**(*message*, *encoding*)

Decodes complete ASTM message that is sent or received due communication routines. It should contains checksum that would be additionally verified.

Parameters

- **message** (*bytes*) – ASTM message.
- **encoding** (*str*) – Data encoding.

Returns

Tuple of three elements:

- int frame sequence number.
- list of records with unicode data.
- bytes checksum.

Raises

- `ValueError` if ASTM message is malformed.
- `AssertionError` if checksum verification fails.

`astm.codec.`**`decode_record`**(*record*, *encoding*)
Decodes ASTM record message.

`astm.codec.`**`decode_repeated_component`**(*component*, *encoding*)
Decodes ASTM field repeated component.

`astm.codec.`**`encode`**(*records*, *encoding='latin-1'*, *size=None*, *seq=1*)
Encodes list of records into single ASTM message, also called as "packed" message.

If you need to get each record as standalone message use *`iter_encode()`* instead.

If the result message is too large (greater than specified *size* if it's not `None`), than it will be split by chunks.

Parameters

- **`records`** (*list*) – List of ASTM records.
- **`encoding`** (*str*) – Data encoding.
- **`size`** (*int*) – Chunk size in bytes.
- **`seq`** (*int*) – Frame start sequence number.

Returns List of ASTM message chunks.

Return type list

`astm.codec.`**`encode_component`**(*component*, *encoding*)
Encodes ASTM record field components.

`astm.codec.`**`encode_message`**(*seq*, *records*, *encoding*)
Encodes ASTM message.

Parameters

- **`seq`** (*int*) – Frame sequence number.
- **`records`** (*list*) – List of ASTM records.
- **`encoding`** (*str*) – Data encoding.

Returns ASTM complete message with checksum and other control characters.

Return type str

`astm.codec.`**`encode_record`**(*record*, *encoding*)
Encodes single ASTM record.

Parameters

- **`record`** (*list*) – ASTM record. Each `str`-typed item counted as field value, one level nested `list` counted as components and second leveled - as repeated components.
- **`encoding`** (*str*) – Data encoding.

Returns Encoded ASTM record.

Return type str

`astm.codec.`**`encode_repeated_component`**(*components*, *encoding*)
Encodes repeated components.

`astm.codec.`**`is_chunked_message`**(*message*)
Checks plain message for chunked byte.

`astm.codec.`**`iter_encode`**(*records*, *encoding='latin-1'*, *size=None*, *seq=1*)

Encodes and emits each record as separate message.

If the result message is too large (greater than specified *size* if it's not `None`), than it will be split by chunks.

Yields ASTM message chunks.

Return type str

`astm.codec.`**`join`**(*chunks*)

Merges ASTM message *chunks* into single message.

Parameters **`chunks`** (*iterable*) – List of chunks as *bytes*.

`astm.codec.`**`make_checksum`**(*message*)

Calculates checksum for specified message.

Parameters **`message`** (*bytes*) – ASTM message.

Returns Checksum value that is actually byte sized integer in hex base

Return type bytes

`astm.codec.`**`split`**(*msg*, *size*)

Split *msg* into chunks with specified *size*.

Chunk *size* value couldn't be less then 7 since each chunk goes with at least 7 special characters: STX, frame number, ETX or ETB, checksum and message terminator.

Parameters

- **`msg`** (*bytes*) – ASTM message.
- **`size`** (*int*) – Chunk size in bytes.

Yield *bytes*

`astm.mapping` :: Message object mappings

class `astm.mapping.`**`Component`**`(`**args*, ***kwargs*`)`
ASTM component mapping class.

class `astm.mapping.`**`ComponentField`**`(`*mapping*, *name=None*, *default=None*`)`
Mapping field for storing record component.

class `astm.mapping.`**`ConstantField`**`(`*name=None*, *default=None*, *field=<astm.mapping.Field object>*`)`
Mapping field for constant values.

```
>>> class Record(Mapping):
...     type = ConstantField(default='S')
>>> rec = Record()
>>> rec.type
'S'
>>> rec.type = 'W'
Traceback (most recent call last):
    ...
ValueError: Field changing not allowed
```

class `astm.mapping.`**`DateField`**`(`*name=None*, *default=None*, *required=False*, *length=None*`)`
Mapping field for storing date/time values.

class `astm.mapping.`**`DateTimeField`**`(`*name=None*, *default=None*, *required=False*, *length=None*`)`
Mapping field for storing date/time values.

class `astm.mapping.`**`DecimalField`**`(`*name=None*, *default=None*, *required=False*, *length=None*`)`
Mapping field for decimal values.

class `astm.mapping.`**`Field`**`(`*name=None*, *default=None*, *required=False*, *length=None*`)`
Base mapping field class.

class `astm.mapping.`**`IntegerField`**`(`*name=None*, *default=None*, *required=False*, *length=None*`)`
Mapping field for integer values.

class `astm.mapping.`**`NotUsedField`**`(`*name=None*`)`
Mapping field for value that should be used. Acts as placeholder. On attempt to assign something to it raises `UserWarning` and rejects assigned value.

class `astm.mapping.`**`Record`**`(`**args*, ***kwargs*`)`
ASTM record mapping class.

class `astm.mapping.`**`RepeatedComponentField`**`(`*field*, *name=None*, *default=None*`)`
Mapping field for storing list of record components.

class `astm.mapping.`**`SetField`**(*name=None*, *default=None*, *required=False*, *length=None*, *values=None*, *field=<astm.mapping.Field object>*)
Mapping field for predefined set of values.

class `astm.mapping.`**`TextField`**(*name=None*, *default=None*, *required=False*, *length=None*)
Mapping field for string values.

class `astm.mapping.`**`TimeField`**(*name=None*, *default=None*, *required=False*, *length=None*)
Mapping field for storing times.

CHAPTER 4

`astm.records` :: Base ASTM records

Common ASTM records structure.

This module contains base ASTM records mappings with only defined common required fields for most implementations. Others are marked as *`NotUsedField`* and should be defined explicitly for your ASTM realisation.

CHAPTER 5

astm.asynclib :: Asynchronous socket handler

`astm.asynclib.`**`loop`**`(`*timeout=30.0*, *map=None*, *tasks=None*, *count=None*`)`

Enter a polling loop that terminates after count passes or all open channels have been closed. All arguments are optional. The *count* parameter defaults to None, resulting in the loop terminating only when all channels have been closed. The *timeout* argument sets the timeout parameter for the appropriate `select()` or `poll()` call, measured in seconds; the default is 30 seconds. The *use_poll* parameter, if true, indicates that `poll()` should be used in preference to `select()` (the default is `False`).

The *map* parameter is a dictionary whose items are the channels to watch. As channels are closed they are deleted from their map. If *map* is omitted, a global map is used. Channels (instances of `asyncore.dispatcher`, `asynchat.async_chat` and subclasses thereof) can freely be mixed in the map.

class `astm.asynclib.`**`Dispatcher`**`(`*sock=None*, *map=None*`)`

The *`Dispatcher`* class is a thin wrapper around a low-level socket object. To make it more useful, it has a few methods for event-handling which are called from the asynchronous loop. Otherwise, it can be treated as a normal non-blocking socket object.

The firing of low-level events at certain times or in certain connection states tells the asynchronous loop that certain higher-level events have taken place. For example, if we have asked for a socket to connect to another host, we know that the connection has been made when the socket becomes writable for the first time (at this point you know that you may write to it with the expectation of success). The implied higher-level events are:

Event	Description
`handle_connect()`	Implied by the first read or write event
`handle_close()`	Implied by a read event with no data available
`handle_accept()`	Implied by a read event on a listening socket

During asynchronous processing, each mapped channel's *`readable()`* and *`writable()`* methods are used to determine whether the channel's socket should be added to the list of channels `select()`ed or `poll()`ed for read and write events.

`accept``()`

Accept a connection.

The socket must be bound to an address and listening for connections. The return value can be either `None` or a pair `(conn, address)` where *conn* is a *new* socket object usable to send and receive data on the connection, and *address* is the address bound to the socket on the other end of the connection.

When `None` is returned it means the connection didn't take place, in which case the server should just ignore this event and keep listening for further incoming connections.

`bind``(`*address*`)`

Bind the socket to *address*.

The socket must not already be bound. The format of *address* depends on the address family — refer to the `socket` documentation for more information. To mark the socket as re-usable (setting the `SO_REUSEADDR` option), call the *`Dispatcher`* object's `set_reuse_addr()` method.

`close()`
Close the socket.

All future operations on the socket object will fail. The remote end-point will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected.

`connect`(*address*)
As with the normal socket object, *address* is a tuple with the first element the host to connect to, and the second the port number.

`create_socket`(*family*, *type*)
This is identical to the creation of a normal socket, and will use the same options for creation. Refer to the `socket` documentation for information on creating sockets.

`handle_accept()`
Called on listening channels (passive openers) when a connection can be established with a new remote endpoint that has issued a *`connect()`* call for the local endpoint.

`handle_close()`
Called when the socket is closed.

`handle_connect()`
Called when the active opener's socket actually makes a connection. Might send a "welcome" banner, or initiate a protocol negotiation with the remote endpoint, for example.

`handle_error()`
Called when an exception is raised and not otherwise handled. The default version prints a condensed traceback.

`handle_write()`
Called when the asynchronous loop detects that a writable socket can be written. Often this method will implement the necessary buffering for performance. For example:

```
def handle_write(self):
    sent = self.send(self.buffer)
    self.buffer = self.buffer[sent:]
```

`listen`(*num*)
Listen for connections made to the socket.

The *num* argument specifies the maximum number of queued connections and should be at least 1; the maximum value is system-dependent (usually 5).

`readable()`
Called each time around the asynchronous loop to determine whether a channel's socket should be added to the list on which read events can occur. The default method simply returns `True`, indicating that by default, all channels will be interested in read events.

`recv`(*buffer_size*)
Read at most *buffer_size* bytes from the socket's remote end-point.

An empty string implies that the channel has been closed from the other end.

`send`(*data*)
Send *data* to the remote end-point of the socket.

`writable()`
Called each time around the asynchronous loop to determine whether a channel's socket should be added

to the list on which write events can occur. The default method simply returns `True`, indicating that by default, all channels will be interested in write events.

class `astm.asyncclib.`**`AsyncChat`**(*sock=None*, *map=None*)

This class is an abstract subclass of *`Dispatcher`*. To make practical use of the code you must subclass *`AsyncChat`*, providing meaningful meth:*found_terminator* method. The *`Dispatcher`* methods can be used, although not all make sense in a message/response context.

Like *`Dispatcher`*, *`AsyncChat`* defines a set of events that are generated by an analysis of socket conditions after a `select()` call. Once the polling loop has been started the *`AsyncChat`* object's methods are called by the event-processing framework with no action on the part of the programmer.

`close_when_done`()

Automatically close this channel once the outgoing queue is empty.

`discard_buffers`()

In emergencies this method will discard any data held in the input and output buffers.

`flush`()

Sends all data from outgoing queue.

`found_terminator`()

Called when the incoming data stream matches the `termination` condition. The default method, which must be overridden, raises a `NotImplementedError` exception. The buffered input data should be available via an instance attribute.

`pull`(*data*)

Puts *data* into incoming queue. Also available by alias *collect_incoming_data*.

`push`(*data*)

Pushes data on to the channel's fifo to ensure its transmission. This is all you need to do to have the channel write the data out to the network.

`readable`()

Predicate for inclusion in the readable for select()

`writable`()

Predicate for inclusion in the writable for select()

CHAPTER 6

ASTM protocol implementation

6.1 astm.protocol :: Common protocol routines

class astm.protocol.**ASTMProtocol**(*sock=None*, *map=None*, *timeout=None*)

Common ASTM protocol routines.

dispatch(*data*)

Dispatcher of received data.

on_ack()

Calls on <ACK> message receiving.

on_enq()

Calls on <ENQ> message receiving.

on_eot()

Calls on <EOT> message receiving.

on_message()

Calls on ASTM message receiving.

on_nak()

Calls on <NAK> message receiving.

on_timeout()

Calls when timeout event occurs. Used to limit waiting time for response data.

6.2 astm.server :: ASTM Server

class astm.server.**BaseRecordsDispatcher**(*encoding=None*)

Abstract dispatcher of received ASTM records by *RequestHandler*. You need to override his handlers or extend dispatcher for your needs. For instance:

```
class Dispatcher(BaseRecordsDispatcher):

    def __init__(self, encoding=None):
        super(Dispatcher, self).__init__(encoding)
        # extend it for your needs
        self.dispatch['M'] = self.my_handler
        # map custom wrappers for ASTM records to their type if you
        # don't like to work with raw data.
        self.wrapper['M'] = MyWrapper
    def on_header(self, record):
        # initialize state for this session
        ...

    def on_patient(self, record):
        # handle patient info
        ...

    # etc handlers

    def my_handler(self, record):
        # handle custom record that wasn't implemented yet by
        # python-astm due to some reasons
        ...
```

After defining our dispatcher, we left only to let `Server` use it:

```
server = Server(dispatcher=Dispatcher)
```

`on_comment`(*record*)
Comment record handler.

`on_header`(*record*)
Header record handler.

`on_order`(*record*)
Order record handler.

`on_patient`(*record*)
Patient record handler.

`on_result`(*record*)
Result record handler.

`on_terminator`(*record*)
Terminator record handler.

`on_unknown`(*record*)
Fallback handler for dispatcher.

class `astm.server.`**`RequestHandler`**(*sock*, *dispatcher*, *timeout=None*)
ASTM protocol request handler.

Parameters

- **`sock`** – Socket object.
- **`dispatcher`** (`BaseRecordsDispatcher`) – Request handler records dispatcher instance.
- **`timeout`** (*int*) – Number of seconds to wait for incoming data before connection closing.

`on_timeout`()
Closes connection on timeout.

class `astm.server.`**`Server`**(*host='localhost'*, *port=15200*, *request=None*, *dispatcher=None*, *timeout=None*, *encoding=None*)
Asyncore driven ASTM server.

Parameters

- **`host`** (*str*) – Server IP address or hostname.

- **port** (*int*) – Server port number.
- **request** – Custom server request handler. If omitted the `RequestHandler` will be used by default.
- **dispatcher** – Custom request handler records dispatcher. If omitted the `BaseRecordsDispatcher` will be used by default.
- **timeout** (*int*) – `RequestHandler` connection timeout. If `None` request handler will wait for data before connection closing.
- **encoding** (*str*) – `Dispatcher`'s encoding.

dispatcher
alias of `BaseRecordsDispatcher`

request
alias of `RequestHandler`

serve_forever (**args*, ***kwargs*)
Enters into the `polling loop` to let server handle incoming requests.

6.3 `astm.client` :: ASTM Client

class `astm.client.`**Client** (*emitter, host='localhost', port=15200, encoding=None, timeout=20, flow_map={'C': ['*'], 'H': ['C', 'P', 'L'], 'L': ['H'], 'O': ['C', 'P', 'O', 'R', 'L'], 'P': ['C', 'O', 'L'], 'R': ['C', 'P', 'O', 'R', 'L'], None: ['H']}, chunk_size=None, bulk_mode=False*)
Common ASTM client implementation.

Parameters

- **emitter** (*function*) – Generator function that will produce ASTM records.
- **host** (*str*) – Server IP address or hostname.
- **port** (*int*) – Server port number.
- **timeout** (*int*) – Time to wait for response from server. If response wasn't received, the `on_timeout()` will be called. If `None` this timer will be disabled.
- **flow_map** – Records flow map. Used by `RecordsStateMachine`.
- **chunk_size** (*int*) – Chunk size in bytes. `None` value prevents records chunking.
- **bulk_mode** (*bool*) – Sends all records for single session (starts from Header and ends with Terminator records) via single message instead of sending each record separately. If result message is too long, it may be split by chunks if *chunk_size* is not `None`. Keep in mind, that collecting all records for single session may take some time and server may reject data by timeout reason.

Type dict

Base *emitter* is a generator that yield ASTM records one by one preserving their order:

```
from astm.records import (
    HeaderRecord, PatientRecord, OrderRecord, TerminatorRecord
)
def emitter():
    assert (yield HeaderRecord()), 'header was rejected'
    ok = yield PatientRecord(name={'last': 'foo', 'first': 'bar'})
    if ok:  # you also can decide what to do in case of record rejection
```

```
        assert (yield OrderRecord())
    yield TerminatorRecord()  # we may do not care about rejection
```

`Client` thought `RecordsStateMachine` keep track on this order, raising `AssertionError` if it is broken.

When *emitter* terminates with `StopIteration` or `GeneratorExit` exception client connection to server closing too. You may provide endless *emitter* by wrapping function body with `while True: ...` loop polling data from source from time to time. Note, that server may have communication timeouts control and may close session after some time of inactivity, so be sure that you're able to send whole session (started by Header record and ended by Terminator one) within limited time frame (commonly 10-15 sec.).

emitter_wrapper
alias of `Emitter`

handle_connect()
Initiates ASTM communication session.

on_ack()
Handles ACK response from server.

Provides callback value `True` to the emitter and sends next message to server.

on_enq()
Raises `NotAccepted` exception.

on_eot()
Raises `NotAccepted` exception.

on_message()
Raises `NotAccepted` exception.

on_nak()
Handles NAK response from server.

If it was received on ENQ request, the client tries to repeat last request for allowed amount of attempts. For others it send callback value `False` to the emitter.

on_timeout()
Sends final EOT message and closes connection after his receiving.

run(*timeout=1.0*, **args*, ***kwargs*)
Enters into the `polling loop` to let client send outgoing requests.

class astm.client.**Emitter**(*emitter*, *flow_map*, *encoding*, *chunk_size=None*, *bulk_mode=False*)
ASTM records emitter for `Client`.

Used as wrapper for user provided one to provide proper routines around for sending Header and Terminator records.

Parameters

- **emitter** – Generator/coroutine.
- **encoding** (*str*) – Data encoding.
- **flow_map** – Records flow map. Used by `RecordsStateMachine`.
- **chunk_size** (*int*) – Chunk size in bytes. If `None`, emitter record wouldn't be split into chunks.
- **bulk_mode** (*bool*) – Sends all records for single session (starts from Header and ends with Terminator records) via single message instead of sending each record separately. If result message is too long, it may be split by chunks if *chunk_size* is not `None`. Keep in mind,

that collecting all records for single session may take some time and server may reject data by timeout reason.

Type dict

close()

Closes the emitter. Acts in same way as `close()` for generators.

send(*value=None*)

Passes *value* to the emitter. Semantically acts in same way as `send()` for generators.

If the emitter has any value within local *buffer* the returned value will be extracted from it unless *value* is `False`.

Parameters **value** (*bool*) – Callback value. `True` indicates that previous record was successfully received and accepted by server, `False` signs about his rejection.

Returns Next record data to send to server.

Return type bytes

state_machine

alias of `RecordsStateMachine`

throw(*exc_type*, *exc_val=None*, *exc_tb=None*)

Raises exception inside the emitter. Acts in same way as `throw()` for generators.

If the emitter had catch an exception and return any record value, it will be proceeded in common way.

CHAPTER 7

ASTM implemetation modules

7.1 `astm.omnilab` :: Omnilab LabOnline

Based on:

file LABONLINE - HOST connection specifications

author Giuseppe Iannucci

revision 2

date 2011-05-31

7.1.1 `astm.omnilab.client` - LabOnline client implementation

class `astm.omnilab.client.`**`Header`**`(`**args*, ***kwargs*`)`

ASTM header record.

Parameters

- **`type`** (*str*) – Record Type ID. Always `H`.
- **`delimeter`** (*str*) – Delimiter Definition. Always `\^&`.
- **`message_id`** (*None*) – Message Control ID. Not used.
- **`password`** (*None*) – Access Password. Not used.
- **`sender`** (`Sender`) – Information about sender. Optional.
- **`address`** (*None*) – Sender Street Address. Not used.
- **`reserved`** (*None*) – Reserved Field. Not used.
- **`phone`** (*None*) – Sender Telephone Number. Not used.
- **`chars`** (*None*) – Sender Characteristics. Not used.
- **`receiver`** (*None*) – Information about receiver. Not used.
- **`comments`** (*None*) – Comments. Not used.
- **`processing_id`** (*str*) – Processing ID. Always `P`.
- **`version`** (*str*) – ASTM Version Number. Always `E 1394-97`.
- **`timestamp`** (*datetime.datetime*) – Date and Time of Message

class `astm.omnilab.client.`**`Patient`**(**args*, ***kwargs*)

ASTM patient record.

Parameters

- **`type`** (*str*) – Record Type ID. Always `P`.
- **`seq`** (*int*) – Sequence Number. Required.
- **`practice_id`** (*str*) – Practice Assigned Patient ID. Required. Length: 12.
- **`laboratory_id`** (*str*) – Laboratory Assigned Patient ID. Required. Length: 16.
- **`id`** (*None*) – Patient ID. Not used.
- **`name`** (`PatientName`) – Patient name.
- **`maiden_name`** (*None*) – Mother's Maiden Name. Not used.
- **`birthdate`** (*datetime.date*) – Birthdate.
- **`sex`** (*str*) – Patient Sex. One of: `M` (male), `F` (female), `I` (animal), `None` is unknown.
- **`race`** (*None*) – Patient Race-Ethnic Origin. Not used.
- **`address`** (*None*) – Patient Address. Not used.
- **`reserved`** (*None*) – Reserved Field. Not used.
- **`phone`** (*None*) – Patient Telephone Number. Not used.
- **`physician_id`** (*None*) – Attending Physician. Not used.
- **`special_1`** (*None*) – Special Field #1. Not used.
- **`special_2`** (*int*) – Patient source. Possible values: - `0`: internal patient; - `1`: external patient.
- **`height`** (*None*) – Patient Height. Not used.
- **`weight`** (*None*) – Patient Weight. Not used.
- **`diagnosis`** (*None*) – Patient's Known Diagnosis. Not used.
- **`medications`** (*None*) – Patient's Active Medications. Not used.
- **`diet`** (*None*) – Patient's Diet. Not used.
- **`practice_1`** (*None*) – Practice Field No. 1. Not used.
- **`practice_2`** (*None*) – Practice Field No. 2. Not used.
- **`admission_date`** (*None*) – Admission/Discharge Dates. Not used.
- **`admission_status`** (*None*) – Admission Status. Not used.
- **`location`** (*str*) – Patient location. Length: 20.
- **`diagnostic_code_nature`** (*None*) – Nature of diagnostic code. Not used.
- **`diagnostic_code`** (*None*) – Diagnostic code. Not used.
- **`religion`** (*None*) – Patient religion. Not used.
- **`martial_status`** (*None*) – Martian status. Not used.
- **`isolation_status`** (*None*) – Isolation status. Not used.
- **`language`** (*None*) – Language. Not used.
- **`hospital_service`** (*None*) – Hospital service. Not used.

- **hospital_institution** (*None*) – Hospital institution. Not used.
- **dosage_category** (*None*) – Dosage category. Not used.

class astm.omnilab.client.**Order** (**args*, ***kwargs*)
ASTM order record.

Parameters

- **type** (*str*) – Record Type ID. Always `O`.
- **seq** (*int*) – Sequence Number. Required.
- **sample_id** (*str*) – Sample ID number. Required. Length: 12.
- **instrument** (*None*) – Instrument specimen ID. Not used.
- **test** (`Test`) – Test information structure (aka Universal Test ID).
- **priority** (*str*) – Priority flag. Required. Possible values: - `S`: stat; -`R`: routine.
- **created_at** (*datetime.datetime*) – Ordered date and time. Required.
- **sampled_at** (*datetime.datetime*) – Specimen collection date and time.
- **collected_at** (*None*) – Collection end time. Not used.
- **volume** (*None*) – Collection volume. Not used.
- **collector** (*None*) – Collector ID. Not used.
- **action_code** (*str*) – Action code. Required. Possible values: - `C`: cancel works for specified tests; - `A`: add tests to existed specimen; - `N`: create new order; - `R`: rerun tests for specified order;
- **danger_code** (*None*) – Danger code. Not used.
- **clinical_info** (*None*) – Revelant clinical info. Not used.
- **delivered_at** (*None*) – Date/time specimen received.
- **biomaterial** (*str*) – Sample material code. Length: 20.
- **physician** (*None*) – Ordering Physician. Not used.
- **physician_phone** (*None*) – Physician's phone number. Not used.
- **user_field_1** (*str*) – An optional field, it will be send back unchanged to the host along with the result. Length: 20.
- **user_field_2** (*str*) – An optional field, it will be send back unchanged to the host along with the result. Length: 1024.
- **laboratory_field_1** (*str*) – In multi-laboratory environment it will be used to indicate which laboratory entering the order. Length: 20.
- **laboratory_field_2** (*str*) – Primary tube code. Length: 12.
- **modified_at** (*None*) – Date and time of last result modification. Not used.
- **instrument_charge** (*None*) – Instrument charge to computer system. Not used.
- **instrument_section** (*None*) – Instrument section id. Not used.
- **report_type** (*str*) – Report type. Always `O` which means normal order request.
- **reserved** (*None*) – Reserved. Not used.
- **location_ward** (*None*) – Location ward of specimen collection. Not used.

- **infection_flag** (*None*) – Nosocomial infection flag. Not used.
- **specimen_service** (*None*) – Specimen service. Not used.
- **laboratory** (*str*) – Production laboratory: in multi-laboratory environment indicates laboratory expected to process the order. Length: 20.

class astm.omnilab.client.**Result**(**args*, ***kwargs*)
ASTM patient record.

Parameters

- **type** (*str*) – Record Type ID. Always `R`.
- **seq** (*int*) – Sequence Number. Required.
- **test** (`Test`) – Test information structure (aka Universal Test ID).
- **value** (*None*) – Measurement value. Numeric, coded or free text value depending on result type. Required. Length: 1024.
- **units** (*None*) – Units. Not used.
- **references** (*None*) – Reference ranges. Not used.
- **abnormal_flag** (*None*) – Result abnormal flag. Not used.
- **abnormality_nature** (*None*) – Nature of abnormality testing. Not used.
- **status** (*None*) – Result status. Not used.
- **normatives_changed_at** (*None*) – Date of changes in instrument normative values or units. Not used.
- **operator** (*None*) – Operator ID. Not used.
- **started_at** (*None*) – When works on test was started on. Not used.
- **completed_at** (*datetime.datetime*) – When works on test was done. Required.
- **instrument** (*None*) – Instrument ID. Not used.

class astm.omnilab.client.**Comment**(**args*, ***kwargs*)
ASTM patient record.

Parameters

- **type** (*str*) – Record Type ID. Always `C`.
- **seq** (*int*) – Sequence Number. Required.
- **source** (*str*) – Comment source. Always `L`.
- **data** (`CommentData`) – Measurement value. Numeric, coded or free text value depending on result type. Required. Length: 1024.
- **ctype** (*str*) – Comment type. Always `G`.

class astm.omnilab.client.**Terminator**(**args*, ***kwargs*)
ASTM terminator record.

Parameters

- **type** (*str*) – Record Type ID. Always `L`.
- **seq** (*int*) – Sequential number. Always `1`.
- **code** (*str*) – Termination code. Always `N`.

7.1.2 `astm.omnilab.server` - LabOnline server implementation

class `astm.omnilab.server.`**`RecordsDispatcher`**`(`**args*, ***kwargs*`)`
Omnilab specific records dispatcher. Automatically wraps records by related mappings.

class `astm.omnilab.server.`**`Header`**`(`**args*, ***kwargs*`)`
ASTM header record.

Parameters

- **`type`** (*str*) – Record Type ID. Always `H`.
- **`delimeter`** (*str*) – Delimiter Definition. Always `\^&`.
- **`message_id`** (*None*) – Message Control ID. Not used.
- **`password`** (*None*) – Access Password. Not used.
- **`sender`** (`Sender`) – Information about sender. Optional.
- **`address`** (*None*) – Sender Street Address. Not used.
- **`reserved`** (*None*) – Reserved Field. Not used.
- **`phone`** (*None*) – Sender Telephone Number. Not used.
- **`chars`** (*None*) – Sender Characteristics. Not used.
- **`receiver`** (*None*) – Information about receiver. Not used.
- **`comments`** (*None*) – Comments. Not used.
- **`processing_id`** (*str*) – Processing ID. Always `P`.
- **`version`** (*str*) – ASTM Version Number. Always `E 1394-97`.
- **`timestamp`** (*datetime.datetime*) – Date and Time of Message

`astm.omnilab.server.`**`Patient`**
alias of `CommonPatient`

class `astm.omnilab.server.`**`Order`**`(`**args*, ***kwargs*`)`
ASTM order record.

Parameters

- **`type`** (*str*) – Record Type ID. Always `O`.
- **`seq`** (*int*) – Sequence Number. Required.
- **`sample_id`** (*str*) – Sample ID number. Required. Length: 12.
- **`instrument`** (`Instrument`) – Instrument specimen ID.
- **`test`** (`Test`) – Test information structure (aka Universal Test ID).
- **`priority`** (*str*) – Priority flag. Required. Possible values: - `S`: stat; -`R`: routine.
- **`created_at`** (*datetime.datetime*) – Ordered date and time. Required.
- **`sampled_at`** (*datetime.datetime*) – Specimen collection date and time.
- **`collected_at`** (*None*) – Collection end time. Not used.
- **`volume`** (*None*) – Collection volume. Not used.
- **`collector`** (*None*) – Collector ID. Not used.

- **action_code** (*str*) – Action code. Required. Possible values: - `None`: normal order result; - `Q`: quality control;
- **danger_code** (*None*) – Danger code. Not used.
- **clinical_info** (*None*) – Revelant clinical info. Not used.
- **delivered_at** (*None*) – Date/time specimen received.
- **biomaterial** (*str*) – Sample material code. Length: 20.
- **physician** (*None*) – Ordering Physician. Not used.
- **physician_phone** (*None*) – Physician's phone number. Not used.
- **user_field_1** (*str*) – An optional field, it will be send back unchanged to the host along with the result. Length: 20.
- **user_field_2** (*str*) – An optional field, it will be send back unchanged to the host along with the result. Length: 1024.
- **laboratory_field_1** (*None*) – Laboratory field #1. Not used.
- **laboratory_field_2** (*str*) – Primary tube code. Length: 12.
- **modified_at** (*None*) – Date and time of last result modification. Not used.
- **instrument_charge** (*None*) – Instrument charge to computer system. Not used.
- **instrument_section** (*None*) – Instrument section id. Not used.
- **report_type** (*str*) – Report type. Always `F` which means final order request.
- **reserved** (*None*) – Reserved. Not used.
- **location_ward** (*None*) – Location ward of specimen collection. Not used.
- **infection_flag** (*None*) – Nosocomial infection flag. Not used.
- **specimen_service** (*None*) – Specimen service. Not used.
- **laboratory** (*None*) – Production laboratory. Not used.

class `astm.omnilab.server.`**`Result`**`(`**args*, ***kwargs*`)`
ASTM patient record.

Parameters

- **type** (*str*) – Record Type ID. Always `R`.
- **seq** (*int*) – Sequence Number. Required.
- **test** (`Test`) – Test information structure (aka Universal Test ID).
- **value** (*None*) – Measurement value. Numeric, coded or free text value depending on result type. Required. Length: 1024.
- **units** (*str*) – Units. Length: 20.
- **references** (*str*) – Normal reference value interval.
- **abnormal_flag** (*str*) – Result abnormal flag. Possible values: - `0`: normal result; - `1`: result out of normal values; - `2`: result out of attention values; - `3`: result out of panic values; +10 Delta-check; +1000 Device alarm. Length: 4.
- **abnormality_nature** (*str*) – Nature of abnormality testing. Possible values: - `N`: normal value; - `L`: below low normal range; - `H`: above high normal range; - `LL`: below low critical range; - `HH`: above high critical range.

- **status** (*str*) – Result status. `F` indicates a final result; `R` indicating rerun. Length: 1.
- **normatives_changed_at** (*None*) – Date of changes in instrument normative values or units. Not used.
- **operator** (`Operator`) – Operator ID.
- **started_at** (*datetime.datetime*) – When works on test was started on.
- **completed_at** (*datetime.datetime*) – When works on test was done.
- **instrument** (`Instrument`) – Instrument ID. Required.

class `astm.omnilab.server.`**`Terminator`**(**args*, ***kwargs*)
ASTM terminator record.

Parameters

- **type** (*str*) – Record Type ID. Always `L`.
- **seq** (*int*) – Sequential number. Always `1`.
- **code** (*str*) – Termination code. Always `N`.

CHAPTER 8

Changelog

8.1 Release 0.5 (2013-03-16)

- Rewrite emitter for Client: replace mystic sessions that hides some detail with explicit need of yielding Header and Terminator records;
- Fix emitter usage with infinity loop;
- Use timer based on scheduled tasks instead of threading.Timer;
- Remove internal states routine;
- Improve overall stability;
- Client now able to send data by chunks and with bulk mode, sending all records with single message;
- Code cleanup;

8.2 Release 0.4.1 (2013-02-01)

- Fix timer for Python 2.x

8.3 Release 0.4 (2013-02-01)

- Fix astm.codec module: now it only decodes bytes to unicode and encodes unicode to bytes;
- Add records dispatcher for server request handler;
- Add session support for astm client emitter;
- Repeat ENQ on timeout;
- Code cleanup and refactoring;
- Set minimal Python version to 2.6, but 3.2-3.3 also works well.

8.4 Release 0.3 (2012-12-15)

- Refactor OmniLab module;

- Rename astm.proto module to astm.protocol;
- Add support for network operation timeouts;
- Client now better communicates with ASTM records emitter;
- Improve logging;
- Code cleanup;
- More tests for base functionality.

8.5 Release 0.2 (2012-12-11)

- Fork, mix and refactor asyncore/asynchat modules to astm.asynclib module which provides more suitable methods to implement asynchronous operations for our task;
- Implement ASTM client and server that handles common protocol routines.

8.6 Release 0.1 (2012-12-09)

- Base decode/encode functions for ASTM data format;
- Small object model mapping based on couchdb-python solution that helps to design records as classes, access to fields by name alias and provide autoconversion between Python objects and ASTM raw values;
- Add demo support of OmniLab LabOnline product.

CHAPTER 9

Licence

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index