pyzm Documentation

Pliable Pixels

Aug 23, 2023

Contents:

Github Repository

CHAPTER 1

pyzm - ZoneMinder API, logger, machine learning and tools for python

1.1 pyzm package

1.1.1 API

1.1.1.1 ZMApi

Python API wrapper for ZM. Exposes login, monitors, events, etc. API

Important: Make sure you have the following settings in ZM:

- `AUTH_RELAY` is set to hashed
- A valid `AUTH_HASH_SECRET` is provided (not empty)
- `AUTH_HASH_IPS` is disabled
- `OPT_USE_APIS` is enabled
- If you are using any version lower than ZM 1.34, `OPT_USE_GOOG_RECAPTCHA` is disabled
- If you are NOT using authentication at all in ZM, that is `OPT_USE_AUTH` is disabled, then make sure you also disable authentication in zmNinja, otherwise it will keep waiting for auth keys.
- I don't quite know why, but on some devices, connection issues are caused because ZoneMinder's CSRF code causes issues. See this thread, for example. In this case, try turning off CSRF checks by going to `ZM->Options->System` and disable "Enable CSRF magic".

---

class pyzm.api.**ZMApi**(*options={}*)

Bases: *pyzm.helpers.Base.Base*

__init__(*options={}*)

Options is a dict with the following keys:

- apiurl - the full API URL (example https://server/zm/api)

- portalurl - the full portal URL (example https://server/zm). Only needed if you are downloading events/images
- user - username (don't specify if no auth)
- password - password (don't specify if no auth)
- disable_ssl_cert_check - if True will let you use self signed certs
- basic_auth_user - basic auth username
- basic_auth_password - basic auth password

Note: you can connect your own customer logging class to the API in which case all modules will use your custom class. Your class will need to implement some methods for this to work. See `pyzm.helpers.Base.SimpleLog` for method details.

authenticated()
True if login API worked

Returns boolean – True if Login API worked

configs(*options={}*)
Returns config values of ZM

Args: options (dict, optional): Defaults to {}. options:

```
{
    'force_reload': boolean # if True, reloads
}
```

Returns ZM configs

Return type `pyzm.helpers.Configs`

events(*options={}*)
Returns list of events based on filter criteria. Note that each time you called events, a new HTTP call is made.

Parameters **options** (*dict, optional*) – Various filters that will be applied to events. Defaults to {}. Available fields:

```
{
    'event_id': string # specific event ID to fetch
    'tz': string # long form timezone (example America/New_York),
    'from': string # string # minimum start time (including human␣
↪readable
                   # strings like '1 hour ago' or '10 minutes ago␣
↪to 5 minutes ago' to create a range)
    'to': string # string # maximum end time
    'mid': int # monitor id
    'min_alarmed_frames': int # minimum alarmed frames
    'max_alarmed_frames': int # maximum alarmed frames
    'object_only': boolean # if True will only pick events
                           # that have objects

}
```

Returns list of events that match criteria

Return type list of *pyzm.helpers.Event*

get_apibase()

get_auth()

get_portalbase()

get_session()

monitors(*options={}*)

Returns list of monitors. Given monitors are fairly static, maintains a cache and returns from cache on subsequent calls.

Args: options (dict, optional): Available fields:

```
{
    'force_reload': boolean # if True refreshes monitors

}
```

Returns list of monitors

Return type list of *pyzm.helpers.Monitor*

restart()

Restarts ZoneMinder

Returns json value of restart command

Return type json

set_state(*state*)

Sets Zoneminder state to specific state

Parameters **state** (*string*) – Name of state

Returns value of state change command

Return type json

start()

Starts ZoneMinder

Returns json value of start command

Return type json

states(*options={}*)

Returns configured states

Parameters **options** (*dict, optional*) – Not used. Defaults to {}.

Returns list of states

Return type list of *pyzm.helpers.State*

stop()

Stops ZoneMinder

Returns json value of stop command

Return type json

tz()

Returns timezone of ZoneMinder server

Returns timezone of ZoneMinder server (or None if API not supported)

Return type string

version()
Returns version of API and ZM

Returns

Version of API and ZM:

```
{
    status: string # if 'error' then will also have 'reason'
    api_version: string # if status is 'ok'
    zm_version: string # if status is 'ok'
}
```

Return type dict

1.1.1.2 Classes returned by the API

Monitors

Holds a list of Monitors for a ZM configuration Given monitors are fairly static, maintains a cache of monitors which can be overriden

class pyzm.helpers.Monitors.**Monitors**(*api=None*)
Bases: *pyzm.helpers.Base.Base*

add(*options={}*)
Adds a new monitor

Parameters **options** (*dict*) – Set of attributes that define the monitor:

```
{
    'function': string # function of monitor
    'name': string # name of monitor
    'enabled': boolean
    'protocol': string
    'host': string
    'port': int
    'path': string
    'width': int
    'height': int
    'raw': {
        # Any other monitor value that is not exposed above.
↪Example:
        'Monitor[Colours]': '4',
        'Monitor[Method]': 'simple'
    }

}
```

Returns json response of API request

Return type json

find(*id=None*, *name=None*)
Given an id or name, returns matching monitor object

Parameters

- **id** (*int, optional*) – MonitorId of monitor. Defaults to None.
- **name** (*string, optional*) – Monitor name of monitor. Defaults to None.

Returns Matching monitor object

Return type *pyzm.helpers.Monitor*

list()

Monitor

Each Monitor will hold a single ZoneMinder Monitor. It is basically a bunch of getters for each access to event data. If you don't see a specific getter, just use the generic get function to get the full object

class pyzm.helpers.Monitor.**Monitor**(*api=None*, *monitor=None*)

Bases: *pyzm.helpers.Base.Base*

arm()

Arms monitor (forces alarm)

Returns API response

Return type json

delete()

Deletes monitor

Returns API response

Return type json

dimensions()

Returns width and height of monitor

Returns

as below:

```
{
    'width': int,
    'height': int
}
```

Return type dict

disarm()

Disarm monito (removes alarm)

Returns API response

Return type json

enabled()

True if monitor is enabled

Returns Enabled or not

Return type bool

eventcount(*options={}*)

Returns count of events for monitor

Parameters **options** (*dict, optional*) – Same as options for *pyzm.helpers.Event*. Defaults to {}.

Returns count

Return type int

events(*options={}*)
Returns events associated to the monitor, subject to filters in options

Parameters **options** (*dict, optional*) – Same as options for *pyzm.helpers.Event*. Defaults to {}.

Returns *pyzm.helpers.Events*

function()
returns monitor function

Returns monitor function

Return type string

get()
Returns monitor object

Returns Monitor object

Return type *pyzm.helpers.Monitor*

id()
Returns monitor Id

Returns Monitor Id

Return type int

name()
Returns monitor name

Returns monitor name

Return type string

set_parameter(*options={}*)
Changes monitor parameters

Parameters **options** (*dict, optional*) – As below. Defaults to {}:

```
{
    'function': string # function of monitor
    'name': string # name of monitor
    'enabled': boolean
    'raw': {
        # Any other monitor value that is not exposed above. Example:
        'Monitor[Colours]': '4',
        'Monitor[Method]': 'simple'
    }

}
```

Returns API Response

Return type json

status()

Returns status of monitor, as reported by zmdc TBD: crappy return, need to normalize

Returns API response

Return type json

type()
Returns monitor type

Returns Monitor type

Return type string

Events

Holds a list of Events for a ZM configuration You can pass different conditions to filter the events Each invocation results in a new API call as events are very dynamic

You are not expected to use this module directly. It is instantiated when you use the *pyzm.api.ZMApi.events()* method of *pyzm.api*

class pyzm.helpers.Events.**Events**(*api=None*, *options=None*)
Bases: *pyzm.helpers.Base.Base*

count()
Returns number of events retrieved in previous invocation

Returns int – number of events

get(*options={}*)
Returns the full list of events. Typically useful if you need access to data for which you don't have an easy getter

Keyword Arguments:

- options: dict with same parameters as the one you pass in *pyzm.api.ZMApi.events()*. This is really a convenience instead of re-creating the instance.

Returns list – of *pyzm.helpers.Events*

list()
Returns list of event

Returns list – events of *Event*

Event

Each Event will hold a single ZoneMinder Event. It is basically a bunch of getters for each access to event data. If you don't see a specific getter, just use the generic get function to get the full object

class pyzm.helpers.Event.**Event**(*event=None*, *api=None*)
Bases: *pyzm.helpers.Base.Base*

alarmed_frames()
Returns total alarmed frames in event.

Returns total alarmed frames

Return type int

cause()
returns event cause.

Returns event cause

Return type string

delete()
Deletes this event

Returns API response

Return type json

download_image(*fid='snapshot'*, *dir='.'*, *show_progress=False*)
Downloads an image frame of the current event object

Parameters

- **fid** (*str, optional*) – Frame ID. Defaults to 'snapshot'.
- **dir** (*str, optional*) – Path to save the image to. Defaults to '.'.
- **show_progress** (*bool, optional*) – If enabled shows a progress bar (if possible). Defaults to False.

Returns path+filename of downloaded image

Return type string

download_video(*dir='.'*, *show_progress=False*)
Downloads a video mp4 of the current event object Only works if there an actual video

Parameters

- **dir** (*str, optional*) – Path to save the image to. Defaults to '.'.
- **show_progress** (*bool, optional*) – If enabled shows a progress bar (if possible). Defaults to False.

Returns path+filename of downloaded video

Return type string

duration()
Returns duration of event in seconds.

Returns duration

Return type float

fspath()
returns the filesystem path where the event is stored. Only available in ZM 1.33+

Returns path

Return type string

get()
Returns event object.

Returns Event object

Return type *pyzm.helpers.Event*

get_image_url(*fid='snapshot'*)
Returns the image URL for the specified frame

Parameters **fid** (*str, optional*) – Default frame identification. Defaults to 'snapshot'.

Returns URL for the image

Return type string

get_video_url()
Returns the video URL for the specified event

Returns URL for the video file

Return type string

id()
returns event id of event.

Returns event id

Return type int

monitor_id()
returns monitor ID of event object.

Returns monitor id

Return type int

name()
returns name of event.

Returns name of event

Return type string

notes()
returns event notes.

Returns event notes

Return type string

score()
Returns total, average and max scores of event.

Returns

As below:

```
{
    'total': float,
    'average': float,
    'max': float
}
```

Return type dict

total_frames()
Returns total frames in event.

Returns total frames

Return type int

video_file()
returns name of video file in which the event was stored.

Returns name of video file

Return type string

States

Holds a list of States for a ZM configuration Given states are fairly static, maintains a cache of states which can be overriden

class pyzm.helpers.States.**States**(*api=None*)
Bases: *pyzm.helpers.Base.Base*

find(*id=None*, *name=None*)
Return a state object that matches either and id or a

Parameters

- **id** (*int, optional*) – Id of state. Defaults to None.
- **name** (*string, optional*) – name of state. Defaults to None.

Returns State object that matches

Return type *pyzm.helpers.State*

list()
Returns list of state objects

Returns list of state objects

Return type list of *pyzm.helpers.State*

State

Each State will hold a single ZoneMinder State. It is basically a bunch of getters for each access to event data. If you don't see a specific getter, just use the generic get function to get the full object

class pyzm.helpers.State.**State**(*api=None*, *state=None*)
Bases: *pyzm.helpers.Base.Base*

active()
whether this state is active or not

Returns True if active

Return type bool

definition()
Returns the description text of this state

Returns description

Return type string

get()
Returns raw state object

Returns raw state object

Return type *pyzm.helpers.State*

id()
Id of this state

Returns id of this state

Return type int

name()
Name of this state

Returns name of this state

Return type string

Base

All classes derive from this Base class. It implements some common functions that apply across all. For now, this basically holds a pointer to the logging function to invoke to log messages including a simple console based print function if none is provided

class pyzm.helpers.Base.**Base**
Bases: object

class pyzm.helpers.Base.**ConsoleLog**
Bases: object

console based logging function that is used if no logging handler is passed

Debug(*level*, *message*, *caller=None*)

Error(*message*, *caller=None*)

Fatal(*message*, *caller=None*)

Info(*message*, *caller=None*)

Panic(*message*, *caller=None*)

Warning(*message*, *caller=None*)

get_level()

set_level(*level*)

1.1.2 Logging

1.1.2.1 ZMLog

Implements a python implementation of ZoneMinder's logging system You could use this to connect it to the APIs if you want

pyzm.ZMLog.**Debug**(*level=1*, *message=None*, *caller=None*)
Debug level ZM message

Parameters

- **level** (*int*) – ZM Debug level
- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**Error**(*message=None*, *caller=None*)
Error level ZM message

Parameters

- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**Fatal**(*message=None*, *caller=None*)
Fatal level ZM message. Quits after logging

Parameters

- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**Info**(*message=None*, *caller=None*)
Info level ZM message

Parameters

- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**Panic**(*message=None*, *caller=None*)
Panic level ZM message. Quits after logging

Parameters

- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**Warning**(*message=None*, *caller=None*)
Warning level ZM message

Parameters

- **message** (*string*) – Message to log
- **caller** (*stack frame info, optional*) – Used to log caller id/line #. Picked up automatically if none. Defaults to None.

pyzm.ZMLog.**close**()
Closes all handles. Invoke this before you exit your app

pyzm.ZMLog.**get_config**()
Returns configuration of ZM logger

Returns config params

Return type dict

pyzm.ZMLog.**init**(*name=None*, *override={}*)
Initializes the ZM logging system. It follows ZM logging principles and ties into appropriate ZM logging levels. Like the rest of ZM, it can write to files, syslog and the ZM DB.

To make it simpler to override, you can pass various options in the override dict. When passed, they will override any ZM setting

Parameters

- **name** (*string, optional*) – Name to be used while writing logs. If not specified, it will use the process name. Defaults to None.

- **override** (*dict, optional*) – Various parameters that can supercede standard ZM logs. Defaults to {}. The parameters that can be overriden are:

```
{
    'dump_console': False,
    'conf_path': '/etc/zm',
    'dbuser' : None,
    'dbpassword' : None,
    'dbhost' : None,
    'webuser': 'www-data',
    'webgroup': 'www-data',
    'dbname' : None,
    'logpath' : '/var/log/zm',
    'log_level_syslog' : 0,
    'log_level_file' : 0,
    'log_level_db' : 0,
    'log_debug' : 0,
    'log_level_debug' : 1,
    'log_debug_target' : '',
    'log_debug_file' :'',
    'server_id': 0,
    'driver': 'mysql+mysqlconnector'
}

You can also pass environment variables (supports .env files too):
- PYZM_CONFPATH : path to zm.conf. Default is /etc/zm. Note that␣
↪if you have a conf file, the values below
                  will automatically be picked up from the conf.␣
↪You can choose to override them by using
                  the variables below

- PYZM_DBUSER: ZM DB user
- PYZM_DBPASSWORD : ZM DB password
- PYZM_DBHOST: ZM DB host
- PYZM_DBNAME  ZM DB Name
- PYZM_WEBUSER: web user
- PYZM_WEBGROUP: web group
- PYZM_LOGPATH: path to ZM logs
- PYZM_SYSLOGLEVEL
- PYZM_FILELOGLEVEL
- PYZM_DBLOGLEVEL
- PYZM_LOGDEBUG
- PYZM_LOGDEBUGLEVEL
- PYZM_LOGDEBUGTARGET
- PYZM_LOGDEBUGFILE
- PYZM_SERVERID

These are specific to pyzm:
- PYZM_DUMPCONSOLE: If True will print logs to terminal
- PYZM_DBDRIVER: Switch the driver pyzm uses to connect to the DB

The order of priority is:

- pyzm_overrides takes priority over env variables
- env variables (supports .env file) take priority over ZM config␣
↪files
- ZM config files is used for whatever is not overriden above
```

pyzm.ZMLog.**set_level**(*level*)

pyzm.ZMLog.**sig_intr**(*sig*, *frame*)

pyzm.ZMLog.**sig_log_rot**(*sig*, *frame*)

1.1.3 Event Notification

1.1.3.1 ZMEventNotification

Implements a python implementation of the ZM ES server.

class pyzm.ZMEventNotification.**ZMEventNotification**(*options*)

__init__(*options*)
Instantiates a thread that connects to the ZM Notification Server

Parameters **options** (*dict*) – As below:

```
{
    'url': string # websocket url
    'user': string # zm user name
    'password': string # zm password
    'allow_untrusted': boolean # set to true for self-signed certs
    'on_es_message': callback function when a message is received
    'on_es_close': callback function when the connection is closed
    'on_es_error': callback function when an error occurs
}
```

Raises ValueError – if no server is provided

connect()
Connect to the ES

disconnect()
Disconnect from the ES

send(*msg*)
Send message to ES

Parameters **msg** (*dict*) – message to send. The message should follow a control structure as specified in The ES developer guide

1.1.4 Memory

1.1.4.1 ZMMemory

Wrapper to access SHM for Monitor status

class pyzm.ZMMemory.**ZMMemory**(*api=None*, *path='/dev/shm'*, *mid=None*)

__init__(*api=None*, *path='/dev/shm'*, *mid=None*)
Initialize self. See help(type(self)) for accurate signature.

alarm_state()
Returns alarm state

Returns

as below:

```
{
    'id': int # state id
    'state': string # name of state
}
```

Return type dict

cause()
Returns alarm and trigger cause as applicable

Returns

as below:

```
{
    'alarm_cause': string
    'trigger_cause': string
}
```

Return type dict

close()
Closes the handle

get()
returns raw shared and trigger data as a dict

Returns

raw shared and trigger data

```
{
    'shared_data': dict, # of shared data,
    'trigger_data': dict # trigger data
}
```

Return type dict

get_shared_data()
Returns just the shared data

Returns shared data

Return type dict

get_trigger_data()
Returns just the trigger data

Returns trigger data

Return type dict

is_alarmed()
True if monitor is currently alarmed

Returns True if monitor is currently alarmed

Return type bool

is_valid()
True if the memory handle is valid

Returns True if memory handle is valid

Return type bool

last_event()
Returns last event ID

Returns last event id

Return type int

reload()
Reloads monitor information. Call after you get an invalid memory report

Raises ValueError – if no monitor is provided

trigger()
Returns trigger information

Returns

as below:

```
{
    'trigger_text': string,
    'trigger_showtext': string,
    'trigger_cause': string,
    'trigger:state' {
        'id': int,
        'state': string
    }
}
```

Return type dict

1.1.5 Machine Learning

1.1.5.1 DetectSequence

Primary entry point to invoke machine learning classes in pyzm It is recommended you only use DetectSequence methods and not lower level interfaces as they may change drastically.

class pyzm.ml.detect_sequence.**DetectSequence**(*options={}*, *global_config={}*)

__init__(*options={}*, *global_config={}*)
Initializes ML entry point with various parameters

Parameters **options** (–) – Variety of ML options. Best described by an example as below

```
options = {
    'general': {
        # sequence of models you want to run for every specified␣
↪frame
        'model_sequence': 'object,face,alpr' ,
        # If 'yes', will not use portalocks
        'disable_locks':'no',

    },
```

(continues on next page)

(continued from previous page)

```
    # We now specify all the config parameters per model_sequence␣
↪entry
    'object': {
        'general':{
            # 'first' - When detecting objects, if there are␣
↪multiple fallbacks, break out
            # the moment we get a match using any object detection␣
↪library.
            # 'most' - run through all libraries, select one that␣
↪has most object matches
            # 'most_unique' - run through all libraries, select one␣
↪that has most unique object matches

            'same_model_sequence_strategy': 'first' # 'first' 'most
↪', 'most_unique', 'union'
            'pattern': '.*' # any pattern
        },

        # within object, this is a sequence of object detection␣
↪libraries. In this case,
        # First try Coral TPU, if it fails try GPU, and finally, if␣
↪configured, try AWS Rekognition
        'sequence': [
        {
            # Intel Coral TPU
            'object_weights':'/var/lib/zmeventnotification/models/
↪coral_edgetpu/ssd_mobilenet_v2_coco_quant_postprocess_edgetpu.
↪tflite',
            'object_labels': '/var/lib/zmeventnotification/models/
↪coral_edgetpu/coco_indexed.names',
            'object_min_confidence': 0.3,
            'object_framework':'coral_edgetpu'
        },
        {
            # YoloV4 on GPU if TPU fails (because sequence strategy␣
↪is 'first')
            'object_config':'/var/lib/zmeventnotification/models/
↪yolov4/yolov4.cfg',
            'object_weights':'/var/lib/zmeventnotification/models/
↪yolov4/yolov4.weights',
            'object_labels': '/var/lib/zmeventnotification/models/
↪yolov4/coco.names',
            'object_min_confidence': 0.3,
            'object_framework':'opencv',
            'object_processor': 'gpu',
            # These are optional below. Default is 416. Change if␣
↪your model is trained for larger sizes
            'model_width': 416,
            'model_height': 416
        },
        {
            # AWS Rekognition object detection
            # More info: https://medium.com/@michael-ludvig/aws-
↪rekognition-support-for-zoneminder-object-detection-40b71f926a80
            'object_framework': 'aws_rekognition'
            'object_min_confidence': 0.7,
            # AWS region unless configured otherwise, e.g. in ~www-
↪data/.aws/config
```

(continues on next page)

(continued from previous page)

```
            'aws_region': 'us-east-1',
            # AWS credentials from /etc/zm/secrets.ini
            # unless running on EC2 instance with instance IAM role␣
↪(which is preferable)
            'aws_access_key_id': '!AWS_ACCESS_KEY_ID',
            'aws_secret_access_key': '!AWS_SECRET_ACCESS_KEY',
            # no other parameters are required
        }
        ]
    },

    # We repeat the same exercise with 'face' as it is next in␣
↪model_sequence
    'face': {
        'general':{
            'same_model_sequence_strategy': 'first'
        },
        'sequence': [{
            # if max_size is specified, not matter what
            # the resize value in stream_options, it will be␣
↪rescaled down to this
            # value if needed
            'max_size':800,
            'face_detection_framework': 'dlib',
            'known_images_path': '/var/lib/zmeventnotification/
↪known_faces',
            'face_model': 'cnn',
            'face_train_model': 'cnn',
            'face_recog_dist_threshold': 0.6,
            'face_num_jitters': 1,
            'face_upsample_times':1
        }]
    },

    # We repeat the same exercise with 'alpr' as it is next in␣
↪model_sequence
    'alpr': {
        'general':{
            'same_model_sequence_strategy': 'first',

            # This can be applied to any model. This means, don't␣
↪run this model
            # unless a previous model detected one of these labels.
            # In this case, I'm not calling ALPR unless we've␣
↪detected a vehile
            # bacause platerec has an API threshold

            'pre_existing_labels':['car', 'motorbike', 'bus', 'truck
↪', 'boat'],

        },
        'sequence': [{
            'alpr_api_type': 'cloud',
            'alpr_service': 'plate_recognizer',
            'alpr_key': g.config['alpr_key'],
            'platrec_stats': 'no',
            'platerec_min_dscore': 0.1,
```

(continues on next page)

(continued from previous page)

```
            'platerec_min_score': 0.2,
        }]
    }
} # ml_options

- global_config (dict): Used by zm_detect and mlapi to pass
  additional config parameters that may not be present in ml_config
```

detect_stream(*stream*, *options={}*, *ml_overrides={}*)

Implements detection on a video stream

Parameters

- **stream** (*string*) – location of media (file, url or event ID)
- **ml_overrides** (*string*) – Ignore it. You will almost never need it. zm_detect uses it for ugly foo
- **options** (*dict, optional*) – Various options that control the detection process. Defaults to {}:
  - delay (int): Delay in seconds before starting media stream
  - delay_between_frames (int): Delay in seconds between each frame read
  - delay_between_snapshots (int): Delay in seconds between each snapshot frame read (useful if you want to read snapshot multiple times, for example. frameset: ['snapshot','snapshot','snapshot'])
  - download (boolean): if True, will download video before analysis. Defaults to False
  - download_dir (string): directory where downloads will be kept (only applies to videos). Default is /tmp
  - start_frame (int): Which frame to start analysis. Default 1.
  - frame_skip: (int): Number of frames to skip in video (example, 3 means process every 3rd frame)
  - max_frames (int): Total number of frames to process before stopping
  - pattern (string): regexp for objects that will be matched. 'frame_strategy' key below will be applied to only objects that match this pattern
  - frame_set (string or list): comma separated frames to read. Example 'alarm,21,31,41,snapshot' or ['snapshot','alarm','1','2'] Note that if you are specifying frame IDs and using ZM, remember that ZM has a frame buffer Default is 20, I think. So you may want to start at frame 21.
  - contig_frames_before_error (int): How many contiguous frames should fail before we give up on reading this stream. Default 5
  - max_attempts (int): Only for ZM indirection. How many times to retry a failed frame get. Default 1
  - sleep_between_attempts (int): Only for ZM indirection. Time to wait before re-trying a failed frame
  - disable_ssl_cert_check (bool): If True (default) will allow self-signed certs to work
  - save_frames (boolean): If True, will save frames used in analysis. Default False

- save_analyzed_frames (boolean): If True, will save analyzed frames (with boxes). Default False
- save_frames_dir (string): Directory to save analyzed frames. Default /tmp
- **frame_strategy: (string): various conditions to stop matching as below**
  - 'most_models': Match the frame that has matched most models (does not include same model alternatives) (Default)
  - 'first': Stop at first match
  - 'most': Match the frame that has the highest number of detected objects
  - 'most_unique' Match the frame that has the highest number of unique detected objects
- resize (int): Width to resize image, default 800
- polygons(object): object # set of polygons that the detected image needs to intersect
- convert_snapshot_to_fid (bool or 'yes'): if True/'yes', will convert 'snapshot' to an actual fid. If you are seeing boxes at wrong places for snapshot frames, this may fix it. However, it can also result in frame 404 errors if that frame ID is not yet written to disk. So you may want to add a delay if you enable this. Default is False.

Returns

representing matched frame, consists of:

- box (array): list of bounding boxes for matched frame
- label (array): list of labels for matched frame
- confidence (array): list of confidences for matched frame
- id (int): frame id of matched frame
- img (cv2 image): image grab of matched frame
- array of objects:
- list of boxes,labels,confidences of all frames matched

Return type

- object

Note:

The same frames are not retrieved depending on whether you set `download` to `True` or `False`. When set to `True`, we use OpenCV's frame reading logic and when `False` we use ZoneMinder's image.php function which uses time based approximation. Therefore, the retrieve different frame offsets, but I assume they should be reasonably close.

get_ml_options()

set_ml_options(*options*, *force_reload=False*)

Use this to change ml options later. Note that models will not be reloaded unless you add force_reload=True

CHAPTER 2

Example Program(s)

2.1 Simple event extraction example

```
import pyzm
import pyzm.api as zmapi
import getpass
import traceback
import pyzm.ZMMemory as zmmemory
import pyzm.helpers.utils as utils
import pyzm.helpers.globals as g



print ('Using pyzm version: {}'.format(pyzm.__version__))

g.logger.set_level(2)

conf = utils.read_config('/etc/zm/secrets.ini')
api_options  = {
    'apiurl': utils.get(key='ZM_API_PORTAL', section='secrets', conf=conf),
    'portalurl':utils.get(key='ZM_PORTAL', section='secrets', conf=conf),
    'user': utils.get(key='ZM_USER', section='secrets', conf=conf),
    #'disable_ssl_cert_check': True
}

zmapi = zmapi.ZMApi(options=api_options)

event_filter = {
    'from': '9 am',
    'to': '7 pm',
    'object_only':False,
    'min_alarmed_frames': 0,
    'max_events':5,
```

(continues on next page)

(continued from previous page)

```
}
cam_events = zmapi.events(event_filter)
print ('I got {} events'.format(len(cam_events.list())))
for e in cam_events.list():
    print ('Event:{} Cause:{} Notes:{}'.format(e.name(), e.cause(), e.notes()))
#cam_events.list()[0].download_image()
```

2.2 Various examples

```
import pyzm
import pyzm.api as zmapi
import getpass
import traceback
import pyzm.ZMMemory as zmmemory
import time
import pyzm.helpers.globals as g

use_zmlog = True
use_zmes = True



has_zmes = False
has_zmlog = False

print ('Using pyzm version: {}'.format(pyzm.__version__))
if use_zmlog:
    try:
        import pyzm.ZMLog as zmlog #only if you want to log to ZM
        has_zmlog = True
    except ImportError as e:
        print ('Could not import ZMLog, function will be disabled:'+str(e))
        zmlog = None


if use_zmes:
    try:
        from pyzm.ZMEventNotification import ZMEventNotification as ZMES
        has_zmes = True
    except ImportError as e:
        print ('Could not import ZMEventNotification, function will be disabled:
↪'+str(e))


def on_es_message(msg):
    print ('======> APP GOT MESSAGE FROM ES: {}'.format(msg))


def on_es_error(err):
    print ('======> APP GOT ERROR  FROM ES: {}'.format(err))
```

(continues on next page)

(continued from previous page)

```
# ----------------- MAIN -------------------------
# Assuming you want to log to ZM
# You can override default ZM Log settings
# programatically
zm_log_override = {
    'log_level_syslog' : 3,
    'log_level_db': -5,
    'log_debug': 1,
    #'log_level_file': -5,
    'log_debug_target': None
}

if has_zmlog:
    zmlog.init(name='apitest',override=zm_log_override)
    print ("Log inited")


i = input ('Try machine learning tests? [y/N]').lower()
if i == 'y':

    import cv2
    fname = input ('Enter full path to image file:')
    if fname:
        print ('Reading {}'.format(fname))
        img = cv2.imread(fname)
    else:
        print ('Reading /tmp/image.jpg')
        img = cv2.imread('/tmp/image.jpg')

    i = input ('Try TPU tests? [y/N]').lower()
    if i == 'y':
        options = {
            'object_weights':'/var/lib/zmeventnotification/models/coral_edgetpu/ssd_
↪mobilenet_v2_coco_quant_postprocess_edgetpu.tflite',
            'object_labels': '/var/lib/zmeventnotification/models/coral_edgetpu/coco_
↪indexed.names',
            'object_min_confidence': 0.3
        }
        import pyzm.ml.coral_edgetpu as tpu
        m = tpu.Tpu(options=options)
        b,l,c = m.detect(img)
        print (b,l,c)

    i = input ('Try OpenCV tests? [y/N]').lower()
    if i == 'y':
        options = {
            'object_weights':'/var/lib/zmeventnotification/models/yolov4/yolov4.
↪weights',
            'object_labels': '/var/lib/zmeventnotification/models/yolov4/coco.names',
            'object_config': '/var/lib/zmeventnotification/models/yolov4/yolov4.cfg',
            'object_processor': 'cpu',
            'object_min_confidence': 0.3
        }
        import pyzm.ml.yolo as yolo
        m = yolo.Yolo(options=options)
```

(continues on next page)

(continued from previous page)

```
        b,l,c = m.detect(img)
        print (b,l,c)

    i = input ('Try Face recognition tests? [y/N]').lower()
    if i == 'y':
        options = {
            'known_images_path': '/var/lib/zmeventnotification/known_faces',
            'face_recog_dist_threshold':0.6,
            'unknown_face_name':'klingon',
            'save_unknown_faces':'no',
            'save_unknown_faces_leeway_pixels':100,
            'unknown_images_path':'/var/lib/zmeventnotification/unknown_faces',
            'face_detection_framework': 'dlib',
            'face_recognition_framework': 'dlib',


        }
        import pyzm.ml.face as face
        m = face.Face(options=options)
        b,l,c = m.detect(img)
        print (b,l,c)

cam_name='DemoVirtualCam1'
api_options = {
    'apiurl': 'https://demo.zoneminder.com/zm/api',
    'portalurl': 'https://demo.zoneminder.com/zm',
    'user': 'zmuser',
    'password': 'zmpass',
    #'disable_ssl_cert_check': True
}



print ('Running examples on {}'.format(api_options[
    'apiurl'
]))

i = input ('Try monitor shared memory tests? [y/N]').lower()
if i == 'y':
    mid = int(input ('Enter monitor ID:'))
    while True:
        k = 'y'
        try:
            m = zmmemory.ZMMemory(mid=mid)
            break
        except Exception as e:
            print ('Error initing: {}'.format(e))
            k = input ('try again, or \'q\' to quit...')
            if k == 'q': break


    while True and k != 'q':
        if m.is_valid():
            print (m.get())
        else:
            print ('Memory not valid')
            try:
```

(continues on next page)

(continued from previous page)

```
                m.reload()
            except Exception as e:
                print ('Error reloading: {}'.format(e))
        k = input ('Try to read again [\'q\' to quit this test]')

if has_zmes:
    i = input ('Test the Event Server? [y/N]').lower()
    if i=='y':
        ES_URL=None
        ES_USER=None
        ES_PASSWORD=None
        ALLOW_UNTRUSTED=True

        if not ES_URL: ES_URL = input ('Enter ES URL (example wss://foo:9000):')
        if not ES_USER: ES_USER = input ('Enter ES user (example admin):')
        if not ES_PASSWORD: ES_PASSWORD = getpass.getpass('Enter ES password:')

        es = ZMES({
            'url':ES_URL,
            'password': ES_PASSWORD,
            'user': ES_USER,
            'allow_untrusted': ALLOW_UNTRUSTED,
            'on_es_message': on_es_message,
            'on_es_error': on_es_error

        })
        # send a legit command

        print ("Sending a valid login")
        es.send({"event":"control","data":{"type":"filter","monlist":"1,2,5,6,7,8,9,10
↪", "intlist":"0,0,0,0,0,0,0,0"}})
        print ("Sleeping for 3 seconds...")
        time.sleep(3)
        # send a bad command
        print ("Sending an invalid command")
        es.send ('Hi From ES')

        input ('press a key to proceed with the rest...')




# lets init the API
try:

    zmapi = zmapi.ZMApi(options=api_options)
except Exception as e:
    print ('Error: {}'.format(str(e)))
    print(traceback.format_exc())
    exit(1)

# Various getter tests

print ("--------| Getting Monitors |-----------")
ms = zmapi.monitors()
for m in ms.list():
```

(continues on next page)

(continued from previous page)

```
    print ('Name:{} Enabled:{} Type:{} Dims:{}'.format(m.name(), m.enabled(), m.
↪type(), m.dimensions()))
    print (m.status())

print ("--------| Getting Events |-----------")
print ("Getting events across all monitors")
event_filter = {
    'from': '24 hours ago', # this will use localtimezone, use 'tz' for other␣
↪timezones
    'object_only':False,
    'min_alarmed_frames': 1,
    'max_events':5,

}

es = zmapi.events(event_filter)
print ('I got {} events'.format(len(es.list())))

input ("Now revoke the access token in ZM and I'll try the same api again. Press␣
↪ENTER when ready....(only applies to ZM 1.33+)")
es = zmapi.events(event_filter)
print ('repeat API - I got {} events'.format(len(es.list())))
input ('press ENTER to continue')

print ('Getting events for {} with filter: {}'.format(cam_name, event_filter))
cam_events = ms.find(name=cam_name).events(options=event_filter)
for e in cam_events.list():
    print ('Event:{} Cause:{} Notes:{}'.format(e.name(), e.cause(), e.notes()))

print ('Now trying to download an image from the first event')
print(cam_events.list())
if cam_events.list():
    e = cam_events.list()[0]
    print (e.name())
    e.download_image(dir='/tmp')
    e.download_video(dir='/tmp')
else:
    print ('No events found')
print ('Getting event summaries')
m = ms.find(name=cam_name)

# These will use server timezone
print ('Monitor {} has {} events {}'.format(m.name(), m.eventcount(options={'from':'1␣
↪hour ago','tz': zmapi.tz()}), '1 hour ago'))
print ('Monitor {} has {} events {}'.format(m.name(), m.eventcount(options={'from':'1␣
↪day ago','tz': zmapi.tz()}), '1 day ago'))

print ("--------| Getting ZM States |-----------")
states = zmapi.states()
for state in states.list():
    print ('State:{}[{}], active={}, details={}'.format(state.name(), state.id(),␣
↪state.active(), state.definition()))

i = input ('Test Monitor State Change? [y/N]').lower()
if i=='y':
    print ("--------| Setting Monitors |-----------")
    m = ms.find(name=cam_name)
```

(continues on next page)

(continued from previous page)

```
    try:
        old_function = m.function()
        input ('Going to change state of {}[{}] to Monitor from {}'.format(m.name(),m.
↪id(), old_function))
        print (m.set_parameter(options={'function':'Monitor'}))
        input ('Switching back to {}'.format(old_function))
        print (m.set_parameter(options={'function':old_function}))
    except Exception as e:
        print ('Error: {}'.format(str(e)))

    print ("--------| Setting Alarms |-----------")
    try:
        input ('Arming {}, press enter'.format(m.name()))
        print (m.arm())
        input ('Disarming {}, press enter'.format(m.name()))
        print (m.disarm())
    except Exception as e:
        print ('Error: {}'.format(str(e)))

i = input ('Test ZM State Changes? [y/N]').lower()
if i=='y':
    print ("--------| Setting States |-----------")
    try:
        input ('Stopping ZM press enter')
        print (zmapi.stop())
        input ('Starting ZM press enter')
        print (zmapi.start())
        for idx,state in enumerate(states.list()):
            print ('{}:{}'.format(idx,state.name()))
        i=int(input('enter state number to switch to:'))
        name = states.list()[i].name()
        print ('Changing state to: {}'.format(name))
        print (zmapi.set_state(state=name))
    except Exception as e:
        print ('Error: {}'.format(str(e)))


print ("--------| Configs Test |-----------")
try:
    conf = zmapi.configs()
    print (conf.find(name='ZM_AUTH_HASH_LOGINS'))
except Exception as e:
    print ('Error: {}'.format(str(e)))

zmlog.close()
```

2.3 Detecting a video stream

```
from pyzm import __version__ as pyzmversion
import pyzm.api as zmapi
import getpass
import traceback
```

(continues on next page)

(continued from previous page)

```
import pyzm.ZMMemory as zmmemory
import time
#import pyzm.ml.object as  ObjectDetect
from pyzm.ml.detect_sequence import DetectSequence
import pyzm.helpers.utils as utils
import sys
import pyzm.helpers.globals as g
import pyzm.ZMLog as log


print ('Using pyzm version: {}'.format(pyzmversion))

#log.init(name='stream', override={'dump_console': True})
g.logger.set_level(5)


#time.sleep(1000)
mid = None

if len(sys.argv) == 1:
    eid = input ('Enter event ID to analyze:')
    mid = input ('Enter MID to use:')
else:
    eid = sys.argv[1]
    if len(sys.argv) == 2:
        print ('Event to analyze:{}'.format(eid))
        mid = input ('Enter MID to use:')
    else:
        mid = sys.argv[2]



'''
api_options = {
    'apiurl': 'https://demo.zoneminder.com/zm/api',
    'portalurl': 'https://demo.zoneminder.com/zm',
    'user': 'zmuser',
    'password': 'zmpass',
    #'disable_ssl_cert_check': True
}
'''

conf = utils.read_config('/etc/zm/secrets.ini')
api_options  = {
    'apiurl': utils.get(key='ZM_API_PORTAL', section='secrets', conf=conf),
    'portalurl':utils.get(key='ZM_PORTAL', section='secrets', conf=conf),
    'user': utils.get(key='ZM_USER', section='secrets', conf=conf),
    'password': utils.get(key='ZM_PASSWORD', section='secrets', conf=conf),
   # 'basic_auth_user': 'bob',
   # 'basic_auth_password': 'bobs password'
   #'disable_ssl_cert_check': True
}


zmapi = zmapi.ZMApi(options=api_options)
ml_options = {
    'general': {
```

(continues on next page)

(continued from previous page)

```
        'model_sequence': 'object,face,alpr',
        'disable_locks': 'no'

    },

    'object': {
        'general':{
            'pattern':'.*',
            'same_model_sequence_strategy': 'most' # also 'most', 'most_unique's
        },
        'sequence': [{
            #First run on TPU
            'name': 'TPU for object detection', # descriptor (optional)
            'enabled': 'no', # skips TPU. Easy way to keep configs but not enable
            'object_weights':'/var/lib/zmeventnotification/models/coral_edgetpu/ssd_
↪mobilenet_v2_coco_quant_postprocess_edgetpu.tflite',
            'object_labels': '/var/lib/zmeventnotification/models/coral_edgetpu/coco_
↪indexed.names',
            'object_min_confidence': 0.3,
            'object_framework':'coral_edgetpu'
        },
        {
            # YoloV4 on GPU if TPU fails (because sequence strategy is 'first')
             'name': 'GPU Yolov4 for object detection', # descriptor (optional)
            'enabled': 'no', # skips. Easy way to keep configs but not enable
            'object_config':'/var/lib/zmeventnotification/models/yolov4/yolov4.cfg',
            'object_weights':'/var/lib/zmeventnotification/models/yolov4/yolov4.
↪weights',
            'object_labels': '/var/lib/zmeventnotification/models/yolov4/coco.names',
            'object_min_confidence': 0.3,
            'object_framework':'opencv',
            'object_processor': 'gpu',
            #'car_past_det_max_diff_area': '10%',
            #'match_past_detections': 'yes',
            #'car_max_detection_size': '13000',
            #'truck_max_detection_size': '13000',
            'image_path': '/var/lib/zmeventnotification/images',

            #'model_width': 512,
            #'model_height': 512
        }]
    },
    'face': {
        'general':{
            'pattern': '.*',
            'same_model_sequence_strategy': 'union'
        },
        'sequence': [{
            'name': 'DLIB face recognition',
            'enabled': 'yes',
            'face_detection_framework': 'dlib',
            'known_images_path': '/var/lib/zmeventnotification/known_faces',
            'face_model': 'cnn',
            'face_train_model': 'cnn',
            'face_recog_dist_threshold': 0.6,
            'face_num_jitters': 1,
            'face_upsample_times':1,
```

(continues on next page)

(continued from previous page)

```
            'max_size': 800
        },
        {
            'name': 'TPU face detection',
            'enabled': 'yes',
            'face_detection_framework': 'tpu',
            'face_weights':'/var/lib/zmeventnotification/models/coral_edgetpu/ssd_
↪mobilenet_v2_face_quant_postprocess_edgetpu.tflite',

            'face_min_confidence': 0.3,

        }]
    },

    'alpr': {
         'general':{
            'same_model_sequence_strategy': 'first',
            'pre_existing_labels':['car', 'motorbike', 'bus', 'truck', 'boat'],

        },
         'sequence': [{
            'alpr_api_type': 'cloud',
            'alpr_service': 'plate_recognizer',
            'alpr_key': utils.get(key='PLATEREC_ALPR_KEY', section='secrets',␣
↪conf=conf),
            'platrec_stats': 'no',
            'platerec_min_dscore': 0.1,
            'platerec_min_score': 0.2,
         }]
    }
} # ml_options

stream_options = {
        #'frame_skip':2,
        #'start_frame': 21,
        #'max_frames':20,
        'strategy': 'most_models',
        #'strategy': 'first',
        'api': zmapi,
        'download': False,
        'frame_set': 'snapshot,alarm',
        'resize': 800,
        'save_frames': False,
        'save_analyzed_frames': False,
        'save_frames_dir': '/tmp',
        'contig_frames_before_error': 5,
        'max_attempts': 3,
        'sleep_between_attempts': 4,
        'disable_ssl_cert_check': True,
        'mid':mid
}


#input ('Enter...')
m = DetectSequence(options=ml_options)
#m = ObjectDetect.Object(options=ml_options)
matched_data,all_data = m.detect_stream(stream=eid, options=stream_options)
```

(continues on next page)

(continued from previous page)

```
print('ALL FRAMES: {}\n\n'.format(all_data))
print ('SELECTED FRAME: {}, SIZE: {}  LABELS: {} BOXES:{} CONFIDENCES:{}'.
↪format(matched_data['frame_id'],matched_data['image_dimensions'],matched_data[
↪'labels'],matched_data['boxes'],matched_data['confidences']))
```

CHAPTER 3

Installation

```
sudo -H pip3 install pyzm
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

P

R

S

T

V

W

Z