intelligent$_{t}racker Documentation$

Release 0.1.0

David Toro

Jul 17, 2018

Contents:

CHAPTER 1

Contents

1.1 intelligent_tracker package

1.1.1 Submodules

1.1.2 intelligent_tracker.array_utils module

intelligent_tracker.array_utils.**check_contours**(*ans*, *expected*, *ignore_shape=False*)

check to contours are the same in an strict or lazy way.

Parameters

- **ans** – contours to check
- **expected** – expected contours or ground truth
- **ignore_shape** – True to ignore order, shapes and check by approximation if ans yields similar results as expected. if ignore_shape is a number then it will be the threshold which is 0.1 when ignore_shape = True.

Returns

intelligent_tracker.array_utils.**convert**(*points*, *roll=None*, *_type=<class 'int'>*, *shift=None*, *apply='contours'*)

Convert contour or points.

Parameters

- **points** – contours or points
- **roll** – roll points
- **_type** – convert points to type
- **shift** – shift points to (x, y)
- **apply** – apply operations the combination "contours-points-list-array"

Returns converted points

intelligent_tracker.array_utils.**draw_contour_groups**(*contour_groups*, *shape=None*, *binary=False*)

draw contours in separate colors

Parameters

- **contour_groups** – list of contours
- **shape** – shape to draw on. If None shape is calculated.
- **binary** – True to draw with Ones, False to draw with colors.

Returns image

intelligent_tracker.array_utils.**find_roll_inv**(*ans*, *expected*)

roll an array until it is best matched to an expected array

Parameters

- **ans** – array to roll
- **expected** – array which ans must be rolled to

Returns best result, inversion, roll, alltrue

intelligent_tracker.array_utils.**is_numpy**(*obj*)

returns True if object is obj numpy object

intelligent_tracker.array_utils.**norm_range**(*vec*, *lower=0*, *upper=255*, *type=<class 'int'>*)

clips vector to range [lower, upper] and a tuple with integers

Parameters

- **vec** – vector
- **lower** – 0
- **upper** – 255
- **type** – int

Returns

1.1.3 intelligent_tracker.core module

class intelligent_tracker.core.**Agent**

Bases: *intelligent_tracker.core.Space*

Anything in the World with an individual behaviour which is movable and cannot be in more than one place at a time, that is and Agent. They can be observable and have positions in the Space. From here anything is derived and populated in the world.

compute()

update internal data if necessary

:return True if computed, else false

computed_vis()

static get_bounding_box_from_cnt(*cnt*, *_type=None*)

static get_bounding_box_from_rotated_box(*rotated_box*, *_type=None*)

static get_cnt_from_bounding_box(*bounding_box*, *_type=<class 'numpy.int32'>*)

static get_cnt_from_rotated_box(*rotated_box*, *_type=<class 'numpy.int32'>*)
get a contour of 4 points with format [left-top, right-top, right-bottom, left-bottom] from a rotated box with format (center, size, angle)

static get_rotated_box_from_bounding_box(*bounding_box*, *_type=None*)

static get_rotated_box_from_cnt(*cnt*, *_type=None*)
get a rotated box format (center, size, angle) from a contour of N points.

raw_vis()

class intelligent_tracker.core.**CompleteGroup**(*iterable=None*, *as_parent=False*, *as_contained=False*)

Bases: *intelligent_tracker.core.Group*

Class to create Groups with faster facilities for indexing and retrieving from indexes (faster retrieval) at the expense of slightly slower times when adding Space objects and an slight increase of memory usage. The difference with a pure Group is negligible when managing small amounts of data. Use this class when the focus is manipulating indexes and comparing data withing or among Groups. For intensive adding and removal of objects use a pure Group.

index(*key*)
Get the index of a given entry, raising an IndexError if it's not present. *key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

reverse()

class intelligent_tracker.core.**Group**(*iterable=None*, *as_parent=False*, *as_contained=False*)

Bases: *intelligent_tracker.core.Space*, collections.abc.MutableSet

Create group of objects withing the Space. This group can contain Space objects and organize them hierarchically by assigning them as children, as contained or simply adding them as private objects which cannot be looked up in the Space.

The Group can be seen as an Ordered set that can iterate them as list and retrieve objects by name or reference as in dictionaries.

add(*key*)
Add *key* as an item to this Group, then return the key.

If *key* is already in the Group, does not adds and returns the key

add_as(*key*, *parent=False*, *contained=False*)
add key to the Group as contained or Group as a parent or both

add_as_child(*key*)
assign this Group as parent of key and add it to the Group

add_as_contained(*key*)
assign this Group as container of key and add it to the Group

append(*key*)
Add *key* as an item to this Group, then return the key.

If *key* is already in the Group, does not adds and returns the key

clear()
Remove all items from this Group.

clear_in_space()
clear objects from group, other groups and space

copy()
copy Group contents

discard(*value*)
Remove an element. Do not raise an exception if absent.

The MutableSet mixin uses this to implement the .remove() method, which *does* raise an error when asked to remove a non-existent item.

give_remove_handle(*key*)
give handle to safely remove key from Group. This should be thread safe and even ig Group is on iterations.

index(*key*)
Get the index of a given entry, raising an IndexError if it's not present.

key can be an iterable of entries that is not a string, in which case this returns a list of indices.

pop()
Remove and return the last element from the Group.

Raises KeyError if the Group is empty.

reverse()

update(*sequence*, *as_parent=False*, *as_contained=False*)
Update the Group with the given iterable sequence, then return the returned value by self.add of the last element inserted.

class intelligent_tracker.core.**GroupHandle**(*parent*, *handle*)
Bases: *intelligent_tracker.core.SpaceHandle*

change_name(*old_name*, *new_name*, *obj*)

remove_name(*name*)

class intelligent_tracker.core.**MetaSpace**(*name*, *bases*, *dct*)
Bases: abc.ABCMeta

Meta class for the Space which gives the "physics" behaviour of the Space

class intelligent_tracker.core.**Point**(*x*, *y*, *z*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

class intelligent_tracker.core.**Space**
Bases: object

Anything that is created must have a name attribute and be in the Space

name

class intelligent_tracker.core.**SpaceHandle**(*parent*, *handle*)
Bases: collections.abc.MutableMapping

handle names

change_name(*old_name*, *new_name*, *obj*)

remove_name(*name*)

class intelligent_tracker.core.**TailItem**(*cnt=None*, *rbox=None*, *bbox=None*, *pt=None*)
Bases: object

TailItem(cnt, rbox, bbox, pt) which behaves like cnt

bbox

cnt

cnt_intersect(*cnt*)
test whether internal cnt is intersected with external cnt

Parameters **cnt** – external contour

Returns True if contours intersect, else False

cnt_near(*cnt*, *min_dist=None*)

enclosing_circle()

point_inside(*point*)
test whether point is inside contour

Parameters **point** – point or x-coordinate, y-coordinate

Returns True if inside or the border of contour, else False

point_near(*point*, *min_dist=50*)

pt

rbox

class intelligent_tracker.core.**WeakRefDictionary**(***kw*)
Bases: *intelligent_tracker.core.WeakWatcherDictionary*

Mapping class that references values weakly.

Entries in the dictionary will be discarded when no strong reference to the value exists anymore

copy()

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

values() → an object providing a view on D's values

class intelligent_tracker.core.**WeakWatcher**(*ob*, *callback=None*, *key=None*, *real_data=None*)
Bases: weakref.KeyedRef

real_data

class intelligent_tracker.core.**WeakWatcherDictionary**(***kw*)
Bases: weakref.WeakValueDictionary

Mapping class that references values weakly.

Entries in the dictionary will be discarded when no strong reference to the value exists anymore

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

class intelligent_tracker.core.**WeakWatcherWithData**(*ob*, *callback=None*, *key=None*, *real_data=None*, ***kwargs*)
Bases: *intelligent_tracker.core.WeakWatcher*

intelligent_tracker.core.**deco_name**(*func*, *ismethod=True*)
wrap function to give always 'name' variable

Parameters

- **func** –
- **ismethod** –

Returns

1.1.4 intelligent_tracker.detectors module

class intelligent_tracker.detectors.**ColorDetector**(*color_lower*, *color_upper*)
Bases: *intelligent_tracker.detectors.Detector*

Detect objects by color

detect_raw_objects(*frame*, *mask=None*)
To modify behaviour of detection

filter_bad_raw_objects(*tail_objects*, *frame*, *mask=None*)

get_BGR_color()
return media BGR color from lower and upper HSV ranges

get_HSV_color()
return media HSV color from lower and upper HSV ranges

get_HSV_color_range()
return lower and upper HSV ranges

set_HSV_color_range(*color_lower=None*, *color_upper=None*)
set lower and upper HSV ranges

class intelligent_tracker.detectors.**Detector**
Bases: *intelligent_tracker.core.Space*

Here a Detector creates an Object or Entity from the real world which will have its own behaviour or "personality". This Detector is the one that classifies the objects and finds them in the real world if they are "lost" or they are not in the scenes anymore until they reappear again.

active_objects()

Returns objects that are active regardless if they are tracking

available_detectors = {'colordetector': <class 'intelligent_tracker.detectors.ColorDe

delete_stray_objects()
delete all objects that are missing the correct target

detect_raw_objects(*frame*, *mask=None*)
To modify behaviour of detection

filter_bad_raw_objects(*tail_objects*, *frame*, *mask=None*)

get_BGR_color()
get detector color from Parent detector or randomly generated

classmethod get_detector(*name*)

inactive_objects()

Returns objects that are not active

process_raw_objects(*frame*, *tail_items*, *bad_items*, *mask=None*)
create new object or reuse object from a tail_item

Parameters

- **frame** –
- **tail_items** –
- **mask** –

Returns

classmethod register_detector(*detector_class*, *name=None*)

track_objects(*frame*, *mask=None*)
only update without creating new objects

Parameters

- **frame** –
- **mask** –

Returns

tracked_objects()

Returns objects that are active and are tracking

untracked_objects()

Returns objects that are not active or are not tracking

class intelligent_tracker.detectors.**EyeDetector**
Bases: *intelligent_tracker.detectors.Detector*

detect_raw_objects(*frame*, *mask=None*)
To modify behaviour of detection

class intelligent_tracker.detectors.**FaceDetector**
Bases: *intelligent_tracker.detectors.Detector*

detect_raw_objects(*frame*, *mask=None*)
To modify behaviour of detection

class intelligent_tracker.detectors.**InvariantCascade**(*angles=None*)
Bases: object

reconstruct(*Ai*, *angle*, *bbox*)

transformations(*frame*)

class intelligent_tracker.detectors.**MovementDetector**
Bases: *intelligent_tracker.detectors.Detector*

class intelligent_tracker.detectors.**Object**(*frame*, *parent_detector*, *max_tail_len=30*, *tracker_type='MEDIANFLOW'*, *key_pts=None*, *descriptors=None*, ***kwargs*)
Bases: *intelligent_tracker.core.Agent*

It is any entity in the World that has its own characteristics or features and that can be tracked in the real world.

add_to_tail(**args*, ***kwargs*)
add a tail_item itself or from a contour (cnt), bounding box (bbox) or rotated box (rbox) to the tail. The point (pt) can be specified on creation but not if tail_item is given

Parameters

- **args** – ('cnt', 'rbox', 'bbox', 'pt')
- **kwargs** – ('cnt', 'rbox', 'bbox', 'pt')

Returns tail_item

cnt

cnt_intersect(*cnt*)
test whether last internal cnt from tail is intersected with external cnt

Parameters **cnt** – external contour

Returns True if contours intersect, else False

cnt_near(*cnt*)
test whether last internal cnt from tail is near with external cnt

Parameters **cnt** – external contour

Returns True if contours intersect, else False

dX
get X position

dY
get Y position

dZ
get Z position

direction(*x_axis=('left', 'right')*, *y_axis=('up', 'down')*, *z_axis=('far', 'near')*)
get tracked object direction in a readable form

Parameters

- **x_axis** – names of the extremes in the x axis. ("left", "right")
- **y_axis** – names of the extremes in the y axis. ("up", "down")
- **z_axis** – names of the extremes in the z axis. ("far", "near")

Returns x_axis, y_axis, z_axis directions

draw_circle(*frame*, *color=None*)

draw_stats(*frame*, *position=None*, *fontFace=None*, *fontScale=None*, *color=None*, *thickness=None*, *tag=None*)

Parameters

- **frame** –

- **position** –
- **fontFace** –
- **fontScale** –
- **color** –
- **thickness** –
- **tag** –

Returns

draw_tail (*frame*, *color=None*, *iterate=None*)
Draw object tail on frame

Parameters

- **frame** – frame to draw on
- **color** – color of tail (1x3 array)
- **iterate** – iterate over positions

Returns

get_BGR_color ()
get object color assigned from Detector or randomly generated

in_zones

is_tracking

max_tail_len

point_inside (*point*)
test whether point is inside object in last position

Parameters **point** – point or x-coordinate, y-coordinate

Returns True if inside or in contour, else False

point_near (*point*)
test whether point is near object in last position

Parameters **point** – point or x-coordinate, y-coordinate

Returns True if inside or in contour, else False

position

Returns last point or position

rotated_box

tail_len

update (*frame*, *mask=None*)

update_tracker (*frame*, *mask=None*, *tracker_type=None*, ***kwargs*)

class intelligent_tracker.detectors.**ObjectDetector**
Bases: *intelligent_tracker.detectors.Detector*

class intelligent_tracker.detectors.**PeopleDetector**
Bases: *intelligent_tracker.detectors.Detector*

intelligent_tracker.detectors.**affine**(*phi*, *img*)

Increase robustness to descriptors by calculating other invariant perspectives to image.

Parameters

- **phi** – rotation of image (in degrees)
- **img** – image to find Affine transforms
- **mask** – mask to detect keypoints (it uses default, mask[:] = 255)

Returns skew_img, skew_mask, Ai (invert Affine Transform)

Ai - is an affine transform matrix from skew_img to img

1.1.5 intelligent_tracker.figures module

1.1.6 intelligent_tracker.forms module

1.1.7 intelligent_tracker.geometry module

class intelligent_tracker.geometry.**BasePoly**(*cnt*, *flags*, *port_left*, *port_right*, *id_left*, *id_right*, *start*, *stop*, *key*)

Bases: object

Base Class to provide basic port allocation, inversion and selection of variables transparently while inverting ports.

static adequate_id(*id*)

adequate or normalize id to be used with all Poly objects

apply_on_invert(*parents=None*)

compare_key(*key*)

give_group_id(*group_id*)

recursively give group_id to all the connected Poly objects

give_port_in_index(*index*, *parent*)

give port in position of index

Parameters

- **index** – 1 for right, 0 for left
- **parent** – parent Poly object

give_port_left(*parent*)

safely assign left port to parent

Parameters **parent** – parent Poly object like an Intersection or Polyline

give_port_right(*parent*)

safely assign right port to parent

Parameters **parent** – parent Poly object like an Intersection or Polyline

has_all_points_inside()

returns True if all points in the lines are inside the other object

id_in_ids(*id*)

test whether id is in this Poly object and in which indices

Parameters **id** – id to test

Returns indices where id is in port_ids

id_left

id_right

indexes(*indices=None*, *invert=None*)
generate lines' point indexes

Parameters **invert** – invert generations of points

Returns generator

invert(*parents=None*, *force=False*)
invert all the chain formed from the connections of Poly objects

Parameters

- **parents** – previous parent in the chain. Control variable indicating which Poly object was the caller or the first to call to_invert to end chain.
- **force** –

Returns

lines_points(*invert=None*)
generate points from contours

Parameters **invert** – invert generation

Returns generator

port_left

port_right

ports_used()
returns True if left and right ports are assigned

process_connections(*conns*, *lines*)
Process connections if they are simple from group A to B.

Parameters

- **conns** – list of group A
- **lines** – list of group B

Returns consumed Counts

recurse_left(*parent=None*)
recursively generate points until a round trip is completed

Parameters **parent** – previous parent in the chain. Control variable indicating which Poly object was the caller or the first to call to_invert to end chain.

Returns generator

recurse_right(*parent=None*)
recursively generate points until a round trip is completed

Parameters **parent** – previous parent in the chain. Control variable indicating which Poly object was the caller or the first to call to_invert to end chain.

Returns generator

test_id(*id*, *position*)
test id if is in position left or right of port_ids

Parameters

- **id** – id to test
- **position** – 1 for right, 0 for left

Returns True for found id in position

to_invert(*parents=None*)
Though any Poly is invertible it would result in processing penalties if many Poly objects are connected together and they are inverted. Thus this functions return True if all the chain can be easily inverted or False if not.

Parameters **parents** – previous parent in the chain. Control variable indicating which Poly object was the caller or the first to call to_invert to end chain.

Returns True for easy to invert, False if not

class intelligent_tracker.geometry.**Completeness**
Bases: object

Search space to add BasePoly objects and find associations

add_id(*id*, *item*)

associate(**args*)

create_associations()
associate connections in all references

generate_count_dictionary()
get ordered dictionary of count of associations

generate_incomplete_set()
create a set with all missing ids

items_connections()
iterate over (id, references to connections)

items_counts()
iterate over (id, count)

register(*item*)
register Connection ids

Parameters **item** –

Returns

sub_id(*id*, *item*)

unregister(*item*)
unregister all ids from a Connection

Parameters **item** –

Returns

exception intelligent_tracker.geometry.**IncompleteAssociations**
Bases: Exception

Exception to raise when a connection could not be determined

class intelligent_tracker.geometry.**Interception**(*id_left*, *center*, *id_right*, *key*)
Bases: *intelligent_tracker.geometry.BasePoly*

Represents a Interception

exception intelligent_tracker.geometry.**NotInvertible**
Bases: Exception

Exception to determine if an object is not invertible

class intelligent_tracker.geometry.**PolyLine**(*cnt*, *flags*, *cnt_id*, *start*, *stop*)
Bases: *intelligent_tracker.geometry.BasePoly*

Represents a Polyline

intelligent_tracker.geometry.**bezier**(*point*, *line*, *check=False*)
Apply bezier algorithm to return a value t from 0 to 1 in x and y, that is tx and ty, if point is inside line where t would be the

percentage of the distance from point1 to point2. If point

is outside line then t<0 or t>1. If line is horizontal then ty is not percentage but the distance from the horizontal and conversely if line is vertical then tx is not percentage but the distance from the vertical.

Parameters

- **point** – point
- **line** – (point1, point2)
- **check** – True to check tx and ty and return None if point is not between point1 and point2 in the line.

Returns tx, ty

intelligent_tracker.geometry.**cnt_check_intersection**(*cnt0*, *cnt1*)
check if normal cnt0 and cnt1 intersect

intelligent_tracker.geometry.**cnt_group**(*cnt*, *cnt_cmp*, *check=False*)
compare cnt pertaining points and give the transitions

Parameters

- **cnt** – testing cnt
- **cnt_cmp** – comparing cnt
- **check** – True to return immediately if a point from cnt is found inside cnt_cmp and with the found flag added to the returned values, True for found or False for not found and consequently with all the flags and transitions plus the found flag.

Returns (flags, transitions) where flags are 1 when the points from cnt which are in cnt_cmp, 0 when when in the contour and -1 when they are not inside. The flag is determined by pointPolygonTest. transitions is an list of the indices where a flag changes from outside to inside or in the contour and vice versa. if check is True: (flags, transitions, found)

intelligent_tracker.geometry.**cnt_intersection**(*cnt0*, *cnt1*)
intersect cnt0 with cnt1

intelligent_tracker.geometry.**draw_drawContours**(*img*, *cnt*)
drawing function used to draw cnt

intelligent_tracker.geometry.**draw_fillConvexPoly**(*img*, *cnt*)
Draw contour. It cannot draw all the cnt correctly. For it to be correct it must be convex.

Parameters

- **img** –
- **cnt** –

Returns

intelligent_tracker.geometry.**draw_fillPoly**(*img*, *cnt*)
drawing function used to draw cnt

intelligent_tracker.geometry.**go_around**(*index*, *size*, *negative=False*)
Correct index to infinitely go around an array. This is equivalent to carrected_index = (index % size) but this function offers more control.

Parameters

- **index** – index to correct
- **size** – size of array
- **negative** – True to not correct negative indexes

Returns (flag, carrected_index) flag indicating that index was corrected

intelligent_tracker.geometry.**intersect_ADD**(*img*, *contours*, *function=<function draw_drawContours>*)
Intersect contours by applying ADDING operations and finally thresholding

Parameters

- **img** – initial binary image
- **contours** – contours to overlap
- **function** – drawing function

Returns final binary image, overlapped contours

intelligent_tracker.geometry.**intersect_AND**(*img*, *contours*, *function=<function draw_drawContours>*)
Intersect contours by applying AND operations

Parameters

- **img** – initial binary image
- **contours** – contours to overlap
- **function** – drawing function

Returns final binary image, overlapped contours

intelligent_tracker.geometry.**intersect_analytical**(*contours*)
Intersect contours by applying purely analytic operations. Contrary to the pixel approach this should not consume much memory and it can yield more precise intersections without adding many points but it could take more time for small contours. If resolution is really big it can save memory because it does not produce accordingly big binary images to obtain the intersections.

Parameters **contours** –

Returns overlapped contours

intelligent_tracker.geometry.**line_intersection**(*line1*, *line2*, *check_inside=True*)
Find the intersecting point between to lines

Parameters

- **line1** – (line1_point1, line1_point2)

- **line2** – (line2_point1, line2_point2)
- **check_inside** – if True and the lines do not cross between their points then it is not considered an intersection and None is returned

Returns point

intelligent_tracker.geometry.**mixed_intersections**(*contours*, *method*, *img*)

intelligent_tracker.geometry.**norm_point**(*pt*)
normalize point to a tuple (x,y)

1.1.8 intelligent_tracker.high_objects module

1.1.9 intelligent_tracker.periferials module

exception intelligent_tracker.periferials.**CameraError**
Bases: Exception

intelligent_tracker.periferials.**PiCamera**
alias of *intelligent_tracker.periferials.UnifiedCamera*

class intelligent_tracker.periferials.**PiRGBArray**(*camera*, *size=None*)
Bases: object

Emulate PiRGBArray in a system that does not have PiCamera support with a normal camera input supported by OpenCV

close()

size

truncate(*val=0*)

class intelligent_tracker.periferials.**SyncCameras**(*cameras*, *resolution=None*, *framerate=None*)
Bases: object

Synchronize cameras

add_camera(*camera*)

capture()

capture_continuous()
continuously produce camera feeds

close()

closed()

framerate

remove_camera(*stream*)

resolutions

start()

class intelligent_tracker.periferials.**UnifiedCamera**(*camera_num=None*)
Bases: object

Emulate PiCamera in a system that does not have PiCamera support with a normal cv2.VideoCapture supported by OpenCV

capture(*rawCapture*, *format='jpeg'*, *use_video_port=False*)

capture_continuous(*rawCapture*, *format='jpeg'*, *use_video_port=False*)

close()

closed()

start_preview()

class intelligent_tracker.periferials.**VideoStream**(*src=None*, *usePiCamera=False*, *resolution=(320, 240)*, *framerate=30*, *format='bgr'*, *trigger=None*)

Bases: object

clear_order()

close()

closed()

framerate

get_frame()
safely give frame from latest read

read()

resolution

start()

1.1.10 intelligent_tracker.persistence module

class intelligent_tracker.persistence.**DEVJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

Extended json decoder with support for instance classes with encoding methods.

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)
```

encode(*obj*)
Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

intelligent_tracker.persistence.**get_obj_module_name**(*obj*)

intelligent_tracker.persistence.**get_obj_name**(*obj*)

intelligent_tracker.persistence.**load_configuration**(*path*, *data=None*, *object_hook=<function object_hook_DEVJSONDecoder>*, *cls_enco=<class 'intelligent_tracker.persistence.DEVJSONEncoder'>*, ***kwargs*)
Load dev extended json file with default data if file is not found.

Parameters

- **path** – load and save path
- **data** – any data supported by the extended json format implemented by dev
- **object_hook** – object_hook_DEVJSONDecoder
- **cls_enco** – DEVJSONEncoder
- **kwargs** – additional arguments for json.load

Returns deserialized json data

intelligent_tracker.persistence.**object_hook_DEVJSONDecoder**(*json_object*)
object_hook compatible with DEVJSONEncoder

Parameters **json_object** –

Returns

intelligent_tracker.persistence.**register_json_class**(*class_obj*, *compatibility_name=None*)

intelligent_tracker.persistence.**register_json_watcher**(*class_obj*, *enco=None*, *deco=None*)

intelligent_tracker.persistence.**save_configuration**(*path*, *data*, *indent=2*, *separators=(', ', ': ')*, *cls=<class 'intelligent_tracker.persistence.DEVJSONEncoder'>*, ***kwargs*)
Save dev extended json serialization.

Parameters

- **path** – save path
- **data** – custom data
- **indent** – 2
- **separators** – (‘,’, ‘: ‘)
- **cls** – DEVJSONEncoder
- **kwargs** – additional arguments for json.dump

Returns

1.1.11 Module contents

class intelligent_tracker.**ContextSupport**
Bases: object

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

D

E

F

G

H

I

L

M

T

U

V

W

X

Y

Z