radar-python

Release 0.0.3

Mar 23, 2020

Contents

1 Quick Start **3**

2 Radar Client **5**

3 Documentation **7**
- 3.1 Install . . . 7
- 3.2 Examples . . . 8
- 3.3 Endpoints . . . 16
- 3.4 Models . . . 22
- 3.5 License . . . 26

4 Indices and tables **29**

Python Module Index **31**

Index **33**

Welcome to Radar! You can use Radar to add location context to your apps with just a few lines of code. This library provides convenient access to Radar's APIs from your python applications or command line. https://radar.io/documentation/api

CHAPTER 1

Quick Start

Want to jump right in? Below is a quick overview of how to get started using the radar-python library to power location based applications. Signup for your free account here https://radar.io/signup and grab your API keys to get started.

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# create a geofence
data = {
    "description": "Example Store",
    "type": "circle",
    "coordinates": [-73.98706, 40.7041029],
    "radius": 100,
    "tag": "store",
    "externalId": "123",
}
new_geofence = radar.geofences.create(data=data)

# Geocode an IP address, converting IP address to country, state if available

ip_location = radar.geocode.ip(ip="107.77.199.117")
"""
>>> print(ip_location)
{
  "city": "Atoka",
  "country": "United States",
  "countryCode": "US",
  "countryFlag": "\ud83c\uddfa\ud83c\uddf8",
  "latitude": 34.385929107666016,
  "longitude": -96.12832641601562,
  "meta": {
    "code": 200
```

(continues on next page)

(continued from previous page)

```
  },
  "postalCode": "74525",
  "state": "Oklahoma",
  "stateCode": "OK"
}
"""

# Compare a route by bike vs foot

origin = (40.7041029, -73.98706)
destination = (40.7141029, -73.99706)
routes = radar.route.distance(origin, destination, modes="bike,foot")
"""
>>> print(f"by foot: {routes.foot}\nby bike: {routes.bike}")
by foot: <distance=2.8 km duration=34 mins>
by bike: <distance=3.2 km duration=12 mins>
"""

# Let a user know what hotels are nearby using place search
user_location = (40.7043, -73.9867)
radar.search.places(near=user_location, categories="hotel-lodging")
"""
[
    <Radar Place: _id='5ded545230409c49f439d943' name='1 Hotel Brooklyn Bridge'␣
↪categories=['hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559f' name='Dazzler Hotels' categories=[
↪'hotel-lodging', 'inn', 'hotel', 'resort']>,
    <Radar Place: _id='59bf2f8d8be4c5ce9409d9f9' name='Hotel St. George' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='5ded528630409c49f41b36a1' name='Hampton Inn' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559c' name='Hampton Inn Brooklyn Downtown'␣
↪categories=['hotel-lodging', 'hotel', 'inn']>
]
"""
```

CHAPTER 2

Radar Client

class radar.**RadarClient**(*secret_key=None*, *pub_key=None*)

The RadarClient class provides convenient access to Radar's API.

API endpoints with authentication level Publishable are safe to call client-side. You should use your publishable API keys to call these endpoints. Use your Test Publishable key for testing and non-production environments. Use your Live Publishable key for production environments.

API endpoints with authentication level Secret are only safe to call server-side. You should use your secret API keys to call these endpoints. Use your Test Secret key for testing and non-production environments. Use your Live Secret key for production environments. Include your API key in the Authorization header.

Examples

```
>>> from radar import RadarClient
>>> radar = RadarClient(secret_key="sk_test_123")
>>> radar.geofences.list()
```

CHAPTER 3

Documentation

3.1 Install

Radar-python requires Python 3.4 or greater.

Below we assume you have the default Python environment already configured on your computer and you intend to install `radar-python` inside of it. If you want to create and work with Python virtual environments, please follow instructions on venv and virtual environments.

First, make sure you have the latest version of `pip` (the Python package manager) installed. If you do not, refer to the Pip documentation and install `pip` first.

3.1.1 Install the released version

The easiest (and best) way to install radar-python is through pip:

Install the current release of `radar-python` with `pip`:

```
$ pip install radar-python
```

To upgrade to a newer release use the `--upgrade` flag:

```
$ pip install --upgrade radar-python
```

If you do not have permission to install software systemwide, you can install into your user directory using the `--user` flag:

```
$ pip install --user radar-python
```

Alternatively, you can manually download `radar-python` from GitHub or PyPI. To install one of these versions, unpack it and run the following from the top-level source directory using the Terminal:

```
$ pip install .
```

3.1.2 Dependencies

Python 3.4+ is required.

- `requests` - python-requests library handles HTTP requests.

Installing through `pip` takes care of dependencies for you.

3.2 Examples

Contents

- *Quick Start*
- *Initialization*
- *Geofences*
- *Events*
- *Users*
- *Context*
- *Search*
- *Geocode*
- *Route*

3.2.1 Quick Start

Want to jump right in? Below is a quick overview of how to get started using the radar-python library to power location based applications. Signup for your free account here https://radar.io/signup and grab your API keys to get started.

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# create a geofence
data = {
    "description": "Example Store",
    "type": "circle",
    "coordinates": [-73.98706, 40.7041029],
    "radius": 100,
    "tag": "store",
    "externalId": "123",
}
new_geofence = radar.geofences.create(data=data)

# Geocode an IP address, converting IP address to country, state if available

ip_location = radar.geocode.ip(ip="107.77.199.117")
"""
```

(continues on next page)

(continued from previous page)

```
>>> print(ip_location)
{
  "city": "Atoka",
  "country": "United States",
  "countryCode": "US",
  "countryFlag": "\ud83c\uddfa\ud83c\uddf8",
  "latitude": 34.385929107666016,
  "longitude": -96.12832641601562,
  "meta": {
    "code": 200
  },
  "postalCode": "74525",
  "state": "Oklahoma",
  "stateCode": "OK"
}
"""

# Compare a route by bike vs foot

origin = (40.7041029, -73.98706)
destination = (40.7141029, -73.99706)
routes = radar.route.distance(origin, destination, modes="bike,foot")
"""
>>> print(f"by foot: {routes.foot}\nby bike: {routes.bike}")
by foot: <distance=2.8 km duration=34 mins>
by bike: <distance=3.2 km duration=12 mins>
"""

# Let a user know what hotels are nearby using place search
user_location = (40.7043, -73.9867)
radar.search.places(near=user_location, categories="hotel-lodging")
"""
[
    <Radar Place: _id='5ded545230409c49f439d943' name='1 Hotel Brooklyn Bridge'␣
↪categories=['hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559f' name='Dazzler Hotels' categories=[
↪'hotel-lodging', 'inn', 'hotel', 'resort']>,
    <Radar Place: _id='59bf2f8d8be4c5ce9409d9f9' name='Hotel St. George' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='5ded528630409c49f41b36a1' name='Hampton Inn' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559c' name='Hampton Inn Brooklyn Downtown'␣
↪categories=['hotel-lodging', 'hotel', 'inn']>
]
"""
```

3.2.2 Initialization

Everything goes through the radar client, so start by initializing RadarClient with your API keys:

```
>>> from radar import RadarClient
>>> radar = RadarClient('secret')
```

The radar client provides access to all the radar API's such as geofences, places, geocoding, search. Examples for each endpoint are included below.

3.2.3 Geofences

Geofences represent custom regions or places monitored in your project. Depending on your use case, a geofence might represent a retail store, a neighborhood, and so on.

Radar geofencing is more powerful than native iOS or Android geofencing, with cross-platform support for unlimited geofences, polygon geofences, and stop detection.

https://radar.io/documentation/geofences

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# create a geofence
data = {
    "description": "Example Store",
    "type": "circle",
    "coordinates": [-73.98706, 40.7041029],
    "radius": 100,
    "tag": "store",
    "externalId": "123",
}
new_geofence = radar.geofences.create(data=data)

# get a geofence by tag and externalId
geofence = radar.geofences.get(tag="store", externalId="123")
print(geofence)

# list geofences
for geofence in radar.geofences.list():
    print(f"Geofence: {geofence._id} - {geofence.description}")

# list users in a geofence
users_in_geofence = radar.geofences.list_users(tag="store", externalId="123")

# delete a geofence, can call geofence.delete() if it's already been fetched
radar.geofences.delete(tag="store", externalId="123")
```

3.2.4 Events

An event represents a change in user state. Events can be uniquely referenced by Radar `_id`.

https://radar.io/documentation/api#events

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# get an event by id
event = radar.events.get(id="123")
print(event)
```

(continues on next page)

(continued from previous page)

```
# list events
for event in radar.events.list():
    print(f"Event: {event.type} at {event.createdAt}")

# list events from a certain time window
from datetime import datetime, timedelta, time

yesterday = datetime.now() - timedelta(days=1)
yesterday_9am = datetime.combine(yesterday, time(9))
yesterday_11am = datetime.combine(yesterday, time(11))
radar.events.list(createdAfter=yesterday_9am, createdBefore=yesterday_11am)

# verify an event
radar.events.verify(id="123", "accept")
radar.events.verify(id="123", value=1)

# delete an event, can call event.delete() if it's already been fetched
radar.events.delete(id="123")
```

Using this method, authentication happens during then initialization of the object. If the authentication is successful, the retrieved session cookie will be used in future requests. Upon cookie expiration, authentication will happen again transparently.

3.2.5 Users

A user represents a user tracked in your project. Users can be referenced by Radar `_id`, `userId`, or `deviceId`

https://radar.io/documentation/api#users

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# get a user by _id, externalId, or deviceId
user = radar.user.get("123")
print(user)

# list the 50 most recently updated users, and any geofences they're in
for user in radar.users.list(limit=50):
    print(f"User: {user._id}, last updated at {user.updatedAt}")
    for geofence in user.geofences:
        print(f"...in geofence {geofence.description}")

# delete a user, can call user.delete() if it's already been fetched
radar.users.delete("123")
```

3.2.6 Context

Gets context for a location without sending device or user identifiers to the server.

https://radar.io/documentation/api#context

```
from radar import RadarClient

# initialize client with your project's secret key and publishable key
SECRET_KEY = "<YOUR SECRET KEY>"
PUB_KEY = "<YOUR PUB KEY>"
radar = RadarClient(secret_key=SECRET_KEY, pub_key=PUB_KEY)

# Get context for a location without sending device or user identifiers to the server.
coordinates = (40.702640, -73.990810)
context = radar.context.get(coordinates=coordinates)
if "place" in dir(context):
    print(f"Location is at place: {context.place.name}")

print(context)
"""
{
  "live": false,
  "geofences": [],
  "place": {
    "_id": "5dc9ada22004860034be2f80",
    "categories": [
      "food-beverage",
      "cafe",
      "coffee-shop"
    ],
    "chain": {
      "domain": "starbucks.com",
      "name": "Starbucks",
      "slug": "starbucks"
    },
    "location": {
      "coordinates": [
        -73.990924,
        40.702719
      ],
      "type": "Point"
    },
    "name": "Starbucks"
  },
  "country": {
    "_id": "5cf694f66da6a800683f4d71",
    "code": "US",
    "name": "United States",
    "type": "country"
  },
  "state": {
    "_id": "5cf695096da6a800683f4e7f",
    "code": "NY",
    "name": "New York",
    "type": "state"
  }
  "dma": {
    "_id": "5cf695016da6a800683f4e06",
    "code": "501",
    "name": "New York",
    "type": "dma"
  },
```

(continues on next page)

(continued from previous page)

```
  "postalCode": {
    "_id": "5cf695286da6a800683f5911",
    "code": "11201",
    "name": "11201",
    "type": "postalCode"
  },
}
"""
```

3.2.7 Search

https://radar.io/documentation/api#search

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# Search for users near a store to send a promotional offer
store_location = (40.7043, -73.9867)
users = radar.search.users(near=store_location)
for user in users:
    # send a push notification
    pass

# Power a store locator using geofence search
user_location = (40.7043, -73.9867)
nearby_stores = radar.search.geofences(near=user_location, tags="store", limit=10)

# Let a user know what hotels are nearby using place search
radar.search.places(near=user_location, categories="hotel-lodging")
"""
[
    <Radar Place: _id='5ded545230409c49f439d943' name='1 Hotel Brooklyn Bridge'␣
↪categories=['hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559f' name='Dazzler Hotels' categories=[
↪'hotel-lodging', 'inn', 'hotel', 'resort']>,
    <Radar Place: _id='59bf2f8d8be4c5ce9409d9f9' name='Hotel St. George' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='5ded528630409c49f41b36a1' name='Hampton Inn' categories=[
↪'hotel-lodging', 'hotel']>,
    <Radar Place: _id='59c1f5898be4c5ce940b559c' name='Hampton Inn Brooklyn Downtown'␣
↪categories=['hotel-lodging', 'hotel', 'inn']>
]
"""

# Power a destination selector using address autocomplete
radar.search.autocomplete(query="20 jay st", near=user_location)
"""
[
    <Radar Address: latitude=40.703945 longitude=-73.98671 formattedAddress='20 Jay␣
↪Street, Brooklyn, New York, NY 11201 USA'>,
    <Radar Address: latitude=40.717976 longitude=-74.010188 formattedAddress='20 Jay␣
↪St, Manhattan, New York, NY 10013 USA'>,
```

(continues on next page)

(continued from previous page)

```
    <Radar Address: latitude=40.74862 longitude=-74.181978 formattedAddress='20 Jay␣
↪St, Newark, NJ USA'>,
    <Radar Address: latitude=40.923457 longitude=-74.170418 formattedAddress='20 Jay␣
↪Street, Paterson, NJ USA'>,
    <Radar Address: latitude=40.908339 longitude=-74.510157 formattedAddress='20 Jay␣
↪St, Rockaway, NJ USA'>
]
"""
```

3.2.8 Geocode

https://radar.io/documentation/api#geocode

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# Geocodes an address, converting address to coordinates.

address = radar.geocode.forward(query="20 jay st brooklyn")[0]
print(address)
print(f"{address.latitude}, {address.longitude}")
"""
>>> print(address)
{
  "borough": "Brooklyn",
  "city": "New York",
  "confidence": "exact",
  "country": "United States",
  "countryCode": "US",
  "countryFlag": "\ud83c\uddfa\ud83c\uddf8",
  "formattedAddress": "20 Jay Street, Brooklyn, New York, NY 11201 USA",
  "geometry": {
    "coordinates": [
      -73.986675,
      40.704262
    ],
    "type": "Point"
  },
  "latitude": 40.704262,
  "longitude": -73.986675,
  "name": "20 Jay Street",
  "neighborhood": "DUMBO",
  "number": "20",
  "postalCode": "11201",
  "state": "New York",
  "stateCode": "NY",
  "street": "Jay Street"
}
>>> print(f"{address.latitude}, {address.longitude}")
40.704262, -73.986675
"""
```

(continues on next page)

(continued from previous page)

```
# Reverse geocodes a location, converting coordinates to address.

address = radar.geocode.reverse(coordinates=(40.7041895, -73.9867797))[0]
"""
>>> print(address)
{
  "borough": "Brooklyn",
  "city": "New York",
  "country": "United States",
  "countryCode": "US",
  "countryFlag": "\ud83c\uddfa\ud83c\uddf8",
  "distance": 0.015,
  "formattedAddress": "20 Jay St, Brooklyn, New York, NY USA",
  "geometry": {
    "coordinates": [
      -73.986802,
      40.704053
    ],
    "type": "Point"
  },
  "latitude": 40.704053,
  "longitude": -73.986802,
  "name": "20 Jay St",
  "neighborhood": "DUMBO",
  "number": "20",
  "state": "New York",
  "stateCode": "NY",
  "street": "Jay St"
}
>>> print(address.formattedAddress)
20 Jay St, Brooklyn, New York, NY USA
"""


# Geocode an IP address, converting IP address to country, state if available

ip_location = radar.geocode.ip(ip="107.77.199.117")
"""
>>> print(ip_location)
{
  "city": "Atoka",
  "country": "United States",
  "countryCode": "US",
  "countryFlag": "\ud83c\uddfa\ud83c\uddf8",
  "latitude": 34.385929107666016,
  "longitude": -96.12832641601562,
  "meta": {
    "code": 200
  },
  "postalCode": "74525",
  "state": "Oklahoma",
  "stateCode": "OK"
}
"""
```

3.2.9 Route

https://radar.io/documentation/api#route

```
from radar import RadarClient

# initialize client with your project's secret key
SECRET_KEY = "<YOUR SECRET KEY>"
radar = RadarClient(SECRET_KEY)

# Compare a route by bike vs foot

origin = (40.7041029, -73.98706)
destination = (40.7141029, -73.99706)
routes = radar.route.distance(origin, destination, modes="bike,foot")
"""
>>> print(f"by foot: {routes.foot}\nby bike: {routes.bike}")
by foot: <distance=2.8 km duration=34 mins>
by bike: <distance=3.2 km duration=12 mins>
"""

# Whats the quickest way to get from a user's origin to destination?

origin = (40.7041029, -73.98706)
destination = (40.7141029, -73.99706)
routes = radar.route.distance(origin, destination, modes="transit,car,bike,foot")

(quickest_mode, quickest_route) = min(
    [("car", routes.car), ("transit", routes.transit), ("bike", routes.bike)],
    key=lambda route: route[1].duration.value,
)
"""
>>> print(f"quickest route is by {quickest_mode}, which will take {quickest_route.
↪duration.value:.2f} min")
quickest route is by car, which will take 9.57 min
"""
```

3.3 Endpoints

Contents

- *Geofences*
- *Users*
- *Events*
- *Context*
- *Geocode*
- *Search*
- *Route*

3.3.1 Geofences

class radar.endpoints.**Geofences**(*radar*, *requester*)

create(*data={}*)
Creates a geofence.

If a geofence with the specified tag and externalId already exists, the request will fail.

https://radar.io/documentation/api#create-geofence

Returns *Geofence*

delete(*id=None*, *tag=None*, *externalId=None*)
https://radar.io/documentation/api#delete-geofence

get(*id=None*, *tag=None*, *externalId=None*)
Gets a geofence by id or tag and externalId

https://radar.io/documentation/api#get-geofence

Returns *Geofence*

list(*limit=None*, *createdBefore=None*, *createdAfter=None*, *tag=None*)
Lists geofences. Geofences are sorted descending by createdAt

https://radar.io/documentation/api#list-geofences

Parameters

- **limit** (*int, optional (default=100)*) – max number of geofences to return.
- **createdBefore** (*datetime, optional (default=None)*) – pagination cursor.
- **createdAfter** (*datetime, optional (default=None)*) – pagination cursor.
- **tag** (*str, optional (default=None)*) – lists geofences with specified tag.

Returns *list* of *Geofence*

list_users(*id=None*, *tag=None*, *externalId=None*, *limit=None*, *updateBefore=None*, *updatedAfter=None*)
Lists users in a geofence.

The geofence can be uniquely referenced by Radar _id or by tag and externalId. Users are sorted descending by updatedAt.

https://radar.io/documentation/api#list-geofence-users

Returns *list* of *User*

upsert(*tag=None*, *externalId=None*, *data={}*)
Upserts a geofence.

If a geofence with the specified tag and externalId already exists, it will be updated. If not, it will be created.

https://radar.io/documentation/api#upsert-geofence

Returns *Geofence*

3.3.2 Users

class radar.endpoints.**Users**(*radar*, *requester*)

delete(*id=None*, *userId=None*, *deviceId=None*)
Deletes a user. The user can be referenced by Radar _id, userId, or deviceId

https://radar.io/documentation/api#delete-user

get(*id=None*, *userId=None*, *deviceId=None*)
Gets a user. The user can be referenced by Radar _id, userId, or deviceId

https://radar.io/documentation/api#get-user

Returns *User*

list(*limit=None*, *updatedBefore=None*, *updatedAfter=None*)
List users, sorted descending by updatedAt

https://radar.io/documentation/api#list-users

Parameters

- **limit** (*int, optional (default=100)*) – Max number of users to return.
- **updateBefore** (*datetime, optional*) – A cursor for use in pagination. Retrieves users updated before the specified datetime.
- **updatedAfter** (*datetime, optional*) – A cursor for use in pagination. Retrieves users updated after the specified datetime.

Returns *list* of *User*

3.3.3 Events

class radar.endpoints.**Events**(*radar*, *requester*)

delete(*id*)
Deletes an event. The event can be uniquely referenced by Radar _id

https://radar.io/documentation/api#delete-event

get(*id*)
Gets an event. The event can be uniquely referenced by Radar _id

https://radar.io/documentation/api#get-event

Returns *Event*

list(*limit=None*, *createdBefore=None*, *createdAfter=None*)
Lists events. Events are sorted descending by createdAt.

https://radar.io/documentation/api#list-events

Parameters

- **limit** (*int, optional (default=100)*) – Max number of events to return.
- **createdBefore** (*datetime, optional*) – A cursor for use in pagination. Retrieves events created before the specified datetime.

- **createdAfter** (*datetime, optional*) – A cursor for use in pagination. Retrieves events created after the specified datetime.

Returns *list* of *Event*

verify(*id*, *verification=None*, *value=None*, *verifiedPlaceId=None*)

Verifies an event.

Events can be accepted or rejected after user check-ins or other forms of verification. Event verifications will be used to improve the confidence level of future events.

https://radar.io/documentation/api#verify-event

Parameters

- **id** (*str*) – id of the event to verify
- **verification** (*str, optional*) – one of "accept", "reject", "unverify"
- **value** (*int, optional*) – one of 1 (accept), -1 (reject), 0 (unverify)
- **verifiedPlaceId** (*str, optional*) – For user.entered_place events, the ID of the verified place.

Example

```
>>> radar.events.verify('123', 'accept')
>>> radar.events.verify('123', value=1)
```

3.3.4 Context

class radar.endpoints.**Context**(*radar*, *requester*)

get(*coordinates*)

Gets context for a location without sending device or user identifiers to the server.

Parameters **coordinates** (*(latitude, longitude)*) – the coordinates of the location

Examples

```
>>> radar.context.get(coordinates=(40.123, -73.456))
```

Returns object with the radar context for the provided location

Return type *RadarContext*

3.3.5 Geocode

class radar.endpoints.**Geocode**(*radar*, *requester*)

forward(*query*)

Geocodes an address, converting address to coordinates.

https://radar.io/documentation/api#geocode-forward

Parameters **query** (*str*) – The address to geocode.

Returns *list* of *Address*

ip(*ip*)
Geocodes an IP address, converting IP address to country.

https://radar.io/documentation/api#geocode-ip

Parameters **ip** (*str*) – The IP address to geocode.

Returns *list* of *Address*

reverse(*coordinates*)
Reverse geocodes a location, converting coordinates to address.

https://radar.io/documentation/api#geocode-reverse

Parameters **coordinates** (*(latitude, longitude)*) – the coordinates to reverse geocode

Returns *list* of *Address*

3.3.6 Search

class radar.endpoints.**Search**(*radar*, *requester*)

autocomplete(*query*, *near*, *limit=None*)
Autocompletes partial addresses and place names, sorted by relevance.

https://radar.io/documentation/api#search-autocomplete

Parameters

- **query** (*str*) – The partial address or place name to autocomplete.
- **near** (*(latitude,longitude)*) – A location for the search in the format (latitude,longitude).
- **limit** (*int, optional (default=10)*) – The max number of addresses to return. A number between 1 and 100.

Returns *list* of *Address*

Example

```
>>> radar.search.autocomplete(near=(40.7041029,-73.98706), categories="coffee-
↪shop")
[<Radar Place: name='Brooklyn Roasting Company'>,
 <Radar Place: name='Starbucks'>,
 <Radar Place: name="Dunkin'">]
```

geofences(*near*, *tags=None*, *radius=None*, *limit=None*)
Searches for geofences near a location, sorted by distance.

https://radar.io/documentation/api#search-geofences

Parameters

- **near** (*(latitude, longitude)*) – A location for the search in the format (latitude,longitude).

- **tags** (*str, optional*) – The tags to filter. A string, comma-separated.
- **radius** (*int, optional (default=1000)*) – The radius to search, in meters. A number between 100 and 10000.
- **limit** (*int, optional (default=100)*) – The max number of geofences to return. A number between 1 and 1000.

Returns *list* of *Geofence*

places(*near*, *chains=None*, *categories=None*, *groups=None*, *radius=None*, *limit=None*)
Searches for places near a location, sorted by distance.

https://radar.io/documentation/api#search-places

Parameters

- **near** (*(latitude,longitude)*) – A location for the search in the format (latitude,longitude).
- **chains** (*str, optional*) – The chain slugs to filter. A string, comma-separated.
- **categories** (*str, optional*) – The categories to filter. A string, comma-separated.
- **groups** (*str, optional*) – The groups to filter. A string, comma-separated.
- **radius** (*int, optional (default=1000)*) – The radius to search, in meters. A number between 100 and 10000.
- **limit** (*int, optional (default=100)*) – The max number of places to return. A number between 1 and 1000.

Returns *list* of *Place*

Example

```
>>> radar.search.places(query="brooklyn roasting", near=(40.7041029,-73.
↪98706))
```

users(*near*, *radius=None*, *limit=None*)
Searches for users near a location, sorted by distance.

https://radar.io/documentation/api#search-users

Parameters

- **near** (*(latitude, longitude)*) – A location for the search. pair of latitude,longitude.
- **radius** (*int, optional (default=1000)*) – The radius to search, in meters. A number between 100 and 10000.
- **limit** (*int, optional (default=100)*) – The max number of places to return. A number between 1 and 1000.

Returns *list* of *User*

3.3.7 Route

class radar.endpoints.**Route**(*radar*, *requester*)

distance(*origin*, *destination*, *modes*, *units='metric'*)
Calculates the travel distance and duration between two locations.

https://radar.io/documentation/api#route-distance

Parameters

- **origin** (*str*) – The origin. A string in the format latitude,longitude.
- **destination** (*str*) – The destination. A string in the format latitude,longitude.
- **modes** (*str*) – The travel modes. A string, comma-separated, including one or more of foot, bike, car, and transit.
- **units** (*str, optional (default="metric")*) – The distance units. A string, metric or imperial.

Returns *Routes*

3.4 Models

3.4.1 Geofence

class radar.models.geofence.**Geofence**(*radar*, *data={}*)
A geofence represents a custom region or place monitored in your project. Geofences can be uniquely referenced by Radar _id or by tag and externalId.

Parameters

- **_id** (*string*) – The unique ID for the geofence, provided by Radar. An alphanumeric string.
- **createdAt** (*datetime*) – The datetime when the geofence was created.
- **live** (*boolean*) – true if the geofence was created with your live API key, false if the user was created with your test API key.
- **tag** (*string*) – An optional group for the geofence.
- **externalId** (*string*) – An optional external ID for the geofence that maps to your internal database.
- **description** (*string*) – A description for the geofence.
- **type** (*string*) – The type of geofence geometry, either polygon or circle.
- **geometry** (*GeoJSON*) – The geometry of the geofence. Coordinates for type polygon. A calculated polygon approximation for type circle. A Polygon in GeoJSON format.
- **geometryCenter** (*GeoJSON*) – The center of the circle for type circle. The calculated centroid of the polygon for type polygon. A Point in GeoJSON format.
- **geometryRadius** (*number*) – The radius of the circle in meters for type circle.
- **metadata** (*dictionary*) – An optional set of custom key-value pairs for the geofence.
- **userId** (*string*) – An optional user restriction for the geofence. If set, the geofence will only generate events for the specified user. If not set, the geofence will generate events for all users.
- **enabled** (*boolean*) – If true, the geofence will generate events. If false, the geofence will not generate events. Defaults to true.

3.4.2 User

class radar.models.user.**User**(*radar*, *data={}*)

A user represents a user tracked in your project. Users can be referenced by Radar _id, userId, or deviceId.

Parameters

- **_id** (*string*) – A unique ID for the user, provided by Radar. An alphanumeric string.
- **live** (*boolean*) – true if the user was created with your live API key, false if the user was created with your test API key.
- **userId** (*string*) – A stable unique ID for the user.
- **deviceId** (*string*) – A device ID for the user.
- **description** (*string*) – An optional description for the user.
- **metadata** (*dict*) – An optional dictionary of custom metadata for the user.
- **location** (*GeoJSON*) – The user's last known location, a Point in GeoJSON format.
- **locationAccuracy** (*number*) – The accuracy of the user's last known location in meters.
- **foreground** (*boolean*) – true if the user's last known location was updated in the foreground, false if the user's last known location was updated in the background.
- **stopped** (*boolean*) – true if the user's last known location was updated while stopped, false if the user's last known location was updated while moving.
- **deviceType** (*string*) – The user's device type, one of iOS, Android, or Web.
- **updatedAt** (*datetime*) – The datetime when the user's location was last updated.
- **geofences** (*list* of *Geofence*) – An array of the user's last known geofences.
- **place** (*Place*) – When Places is enabled, the user's last known place.
- **insights** (*dict*) – When Insights is enabled, the user's learned approximate home and office locations, and home, office, and traveling state.

3.4.3 Event

class radar.models.event.**Event**(*radar*, *data={}*)

An event represents a change in user state. Events can be uniquely referenced by Radar _id.

Parameters

- **_id** (*str*) – The unique ID for the event, provided by Radar. An alphanumeric string.
- **createdAt** (*datetime*) – The datetime when the event was created.
- **live** (*bool*) – true if the event was generated for a user and geofence created with your live API key, false if the event was generated for a user and geofence created with your test API key.
- **type** (*str*) – The type of event. By default, events are generated when a user enters a geofence (type user.entered_geofence) or exits a geofence (type user.exited_geofence). When Insights is enabled, events will also be generated when a user enters their home (type user.entered_home), exits their home (type user.exited_home), enters their office (type user.entered_office), exits their office (type user.exited_office), starts traveling (type user.started_traveling), or stops traveling (user.stopped_traveling).

- **user** (*dict*) – The user for which the event was generated.
- **geofence** (*dict*) – For user.entered_geofence and user.exited_geofence events, the geofence for which the event was generated, including description, tag, and externalId.
- **place** (*dict*) – For user.entered_place and user.exited_place events, the place for which the event was generated, including name, categories, chain, and facebookId.
- **alternatePlaces** (*list*) – For user.entered_place events, alternate place candidates.
- **verifiedPlace** (*dict*) – For verified user.entered_place events, the verified place.
- **location** (*GeoJSON*) – The location of the user at the time of the event. A Point in GeoJSON format.
- **locationAccuracy** (*float*) – The accuracy of the user's location at the time of the event in meters.
- **confidence** (*int*) – The confidence level of the event, one of 3 (high), 2 (medium), or 1 (low).
- **duration** (*float*) – The duration between entry and exit events, in minutes, for user.exited_geofence and user.exited_place events.
- **verification** (*int*) – The verification for the event, one of 1 (accepted), -1 (rejected), or 0 (unverified).

3.4.4 Place

class radar.models.place.**Place**(*radar*, *data={}*)

Place of interest

Parameters

- **_id** (*str*) –
- **name** (*str*) –
- **chain** (*dict*) –
- **facebookPlaceId?** (*str*) –
- **facebookId?** (*str*) –
- **location** (*GeoJSON.Point*) –
- **categories** (*list* of str) –
- **metadata** (*dict*) –

3.4.5 Context

class radar.models.context.**RadarContext**(*radar*, *data={}*)

Location context

Parameters

- **live** (*bool*) –
- **geofences** (*list* of *Geofence*) –
- **place** (*list* of *Place*, optional) –
- **country** (*Region*, optional) –

- **state** (*Region*, optional) –
- **dma** (*Region*, optional) –
- **postalCode** (*Region*, optional) –
- **fraud** (*FraudObject, optional*) –

3.4.6 Region

class radar.models.region.**Region**(*radar*, *data={}*)

A location region, either country, state, dma, or postalCode

Parameters

- **_id** (*str*) –
- **type** (*str*) –
- **name** (*str*) –
- **code** (*str*) –
- **flag** (*str*) –

3.4.7 Address

class radar.models.address.**Address**(*radar*, *data={}*)

Location address

Parameters

- **borough** (*str*) –
- **city** (*str*) –
- **confidence** (*str*) – one of 'exact', 'interpolated', 'fallback'
- **country** (*str*) –
- **countryCode** (*str*) –
- **countryFlag** (*str*) –
- **distance** (*float*) –
- **formattedAddress** (*str*) –
- **geometry** (*GeoJSON.Point*) –
- **name** (*str*) –
- **number** (*str*) –
- **latitude** (*float*) –
- **longitude** (*float*) –
- **neighborhood** (*str*) –
- **postalCode** (*str*) –
- **source** (*str*) –
- **state** (*str*) –

- **stateCode** (*str*) –
- **street** (*str*) –

3.4.8 Route

class radar.models.route.**Route**(*distance=None*, *duration=None*, *mode=None*)
The travel distance and duration between two locations

Parameters

- **mode** (*str*) – one of 'car', 'bike', 'foot', 'transit'
- **duration** (*RouteDuration*) –
- **distance** (*RouteDistance*) –

Routes

class radar.models.route.**Routes**(*radar*, *data={}*)
A collection of *Route*

RouteDistance

class radar.models.route.**RouteDistance**(*value*, *text*)
Travel distance of the route

Parameters

- **value** (*str*) – value of distance in requested units
- **text** (*str*) – human readable distance

RouteDuration

class radar.models.route.**RouteDuration**(*value*, *text*)
Travel duration of the route

Parameters

- **value** (*str*) – minutes
- **text** (*str*) – human readable duration

3.5 License

```
MIT License

Copyright (c) 2020 Radar

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

(continues on next page)

(continued from previous page)

```
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

radar, 5

Index

A

Address (*class in radar.models.address*), 25
autocomplete() (*radar.endpoints.Search method*), 20

C

Context (*class in radar.endpoints*), 19
create() (*radar.endpoints.Geofences method*), 17

D

delete() (*radar.endpoints.Events method*), 18
delete() (*radar.endpoints.Geofences method*), 17
delete() (*radar.endpoints.Users method*), 18
distance() (*radar.endpoints.Route method*), 21

E

Event (*class in radar.models.event*), 23
Events (*class in radar.endpoints*), 18

F

forward() (*radar.endpoints.Geocode method*), 19

G

Geocode (*class in radar.endpoints*), 19
Geofence (*class in radar.models.geofence*), 22
Geofences (*class in radar.endpoints*), 17
geofences() (*radar.endpoints.Search method*), 20
get() (*radar.endpoints.Context method*), 19
get() (*radar.endpoints.Events method*), 18
get() (*radar.endpoints.Geofences method*), 17
get() (*radar.endpoints.Users method*), 18

I

ip() (*radar.endpoints.Geocode method*), 20

L

list() (*radar.endpoints.Events method*), 18
list() (*radar.endpoints.Geofences method*), 17
list() (*radar.endpoints.Users method*), 18
list_users() (*radar.endpoints.Geofences method*), 17

P

Place (*class in radar.models.place*), 24
places() (*radar.endpoints.Search method*), 21

R

radar (*module*), 5
RadarClient (*class in radar*), 5
RadarContext (*class in radar.models.context*), 24
Region (*class in radar.models.region*), 25
reverse() (*radar.endpoints.Geocode method*), 20
Route (*class in radar.endpoints*), 21
Route (*class in radar.models.route*), 26
RouteDistance (*class in radar.models.route*), 26
RouteDuration (*class in radar.models.route*), 26
Routes (*class in radar.models.route*), 26

S

Search (*class in radar.endpoints*), 20

U

upsert() (*radar.endpoints.Geofences method*), 17
User (*class in radar.models.user*), 23
Users (*class in radar.endpoints*), 18
users() (*radar.endpoints.Search method*), 21

V

verify() (*radar.endpoints.Events method*), 19