WunderWeather Documentation

Release 1.0

Tyler Santos

Apr 06, 2018

Contents

Whether you're already a user and want a refresher on the documentation or you're evaluating the package for the first time, you've come to the right place. So what do you want to learn more about?

CHAPTER 1

Introduction

WunderWeather attempts to expose data supplied by Weather Underground in a way that is easy to use and easy to get weather data into your application quickly without having to deal with all of the details.

1.1 About the Wunderground API

The Wunderground API supplies different endpoints called data features which, when supplied the proper arguments, return numerous data points describing the feature being queried.

Documentation

1.2 About the WunderWeather API

WunderWeather was built to expose the data supplied by Wunderground in a uniform fashion. For certain data features where it applied, wrapper classes were created to normalize the data returned and supply ease of access to that data.

When developing WunderWeather there were a few key concepts kept in mind which are listed below. If you intend on contribu

1. **Out of the hundreds of data points that Wunderground so graciously supplies, expose shortcuts to the more frequent**

(a) For the history data feature, Wunderground exposes the average temperature data point using 3 keys rather than the one abstracted in WunderWeather

Wunderground:

```
>>> response["history"]["daily_summary"]["meantempi"]
```

WunderWeather:

```
>>> response.temp_f
```

2. **Normalize the data point names being exposed.**

   (a) The Wunderground API does a great job at supplying endless amounts of weather data but unfortunately similar data points across different features have different names. A case where this crops up frequently is for imperial (i) and metric (m) and their respective Fahrenheit (f) and Celsius (c) identifiers for temperature.

   Example Data Points:

   - temp_i vs temp_f
   - temp_m vs temp_c

1.3 Installation

```
pip install WunderWeather
```

1.4 Code Examples

The following code snippets are examples of extracting data from data feature responses. Some examples build off of previous examples (as to avoid repetition) but should be properly documented as `continuation from NNN example`.

Warning: The WunderWeather package is only Python 3 compatible.

Note: Because the requests package is awesome, we're going to be using that to make our requests in the following examples. We use it to make requests in our package and so should you!

1.4.1 Weather Underground Example

Not using Requests

Example listed in Wunderground documentation

```
1 import urllib2
2 import json
3 f = urllib2.urlopen('http://api.wunderground.com/api/<YOUR_API_KEY>/geolookup/
  ↪conditions/q/IA/Cedar_Rapids.json')
4 json_string = f.read()
```

```
parsed_json = json.loads(json_string)
location = parsed_json['location']['city']
temp_f = parsed_json['current_observation']['temp_f']
print "Current temperature in %s is: %s" % (location, temp_f)
f.close()
```

example

Using Requests

Example listed in Wunderground doc converted to use requests

```
import requests # learn more: https://python.org/pypi/requests
response = requests.get('http://api.wunderground.com/api/<YOUR_API_KEY>/geolookup/
↪conditions/q/MA/Boston.json').json()
location = response['location']['city']
temp_f = response['current_observation']['temp_f']
print("Current Temperature in %s is: %s" %(location,temp_f))
```

Using WunderWeather

Example listed in Wunderground doc converted to use WunderWeather

```
from WunderWeather import weather
extractor = weather.Extract(api_key)
[location,current] = extractor.features("MA/Boston",(('geolookup',''),('now','')))
print("Current Temperature in %s is: %s" %(location.data.city,current.temp_f))
```

In the example above, notice how data points can be extracted from a feature using dotted notation whether there is a feature specific wrapper class or not to provide a uniform look in the calling application. When referencing shortcuts from wrapper classes or directly accessing the data, the look is the same. As of writing this documentation Geolookup does not have a wrapper so all data extracted from that feature must use the `WeatherBase.data` *member to use the dotted notation.*

1.4.2 Additional Examples

```
From pprint import pprint
import arrow
From WunderWeather import weather

# setup
api_key = "your api key"
location = 'MA/Boston'
extractor = weather.Extract(api_key)

# alerts
response = extractor.alerts(location)
pprint(response.data)

# astronomy
response = extractor.astronomy(location)
pprint(response.data)

# geolookup
response = extractor.geolookup(location)
pprint(response.data)

```

```
# history
date = arrow.get("20170601","YYYYMMDD")
response = extractor.date(location,date.format('YYYYMMDD'))
pprint(response.data)

# addl date detail
for observation in response.observations:
print("Date:",observation.date_pretty)
print("Temp:",observation.temp_f)
```

1.5 Explore the WunderWeather API

1.5.1 WunderWeather

WunderWeather package

Submodules

WunderWeather.date module

class WunderWeather.date.**Date**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one (date) history type data feature response.

observations
Abstract the observations for given date

Notes

In a date based response (History,Yesterday) there is a list of observations.

observations
list – List of dictionaries for each observation for the date

Returns list of Observation instances

temp_c

temp_f

class WunderWeather.date.**Observation**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one date based data feature's observations

date_pretty

temp_c

temp_f

class WunderWeather.date.**Range**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one (date) history type data feature response.

high_avg_temp_c

high_avg_temp_f

low_avg_temp_c

low_avg_temp_f

WunderWeather.forecast module

class WunderWeather.forecast.**Forecast**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one Forecast-y type data feature response.

date

periods
Abstract the periods for given forecast

Notes

In a forecast response there are two list of dictionaries. One list is of text representations for each period and the other list is detailed data of each period. This member joins the two lists together, merging dictionaries of matching period keys. Each list is not always sorted in period increasing order and some periods may be missing from each list.

periods
defaultdict – of period data

txt_periods
dict – list of dicts representing text period data

simple_periods
dict – list of dict representing detailed period data

list_of_periods
str – single list of period dicts for processing

period_dict
dict – One period in a period list being processed

Returns list of Period instances

class WunderWeather.forecast.**Period**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one Period in a forecast

date_pretty

high_temp_f

period

text

text_metric

WunderWeather.test module

class WunderWeather.test.**TestAlerts**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_get_item_count()

test_get_one_item_data()

class WunderWeather.test.**TestAstronomy**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

class WunderWeather.test.**TestDate**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

test_shorthand_child_based_get()

class WunderWeather.test.**TestDaycast**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

test_shorthand_child_based_get()

class WunderWeather.test.**TestGeolookup**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

class WunderWeather.test.**TestHourlyDaycast**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_get_item_count()

test_get_one_item_data()

class WunderWeather.test.**TestHurricane**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

class WunderWeather.test.**TestRange**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

class WunderWeather.test.**TestRawtide**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_get_item_count()

test_get_one_item_data()

class WunderWeather.test.**TestTide**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_get_item_count()

test_get_one_item_data()

class WunderWeather.test.**TestTodayHistorical**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

class WunderWeather.test.**TestTodayNow**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

class WunderWeather.test.**TestWeekcast**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

test_shorthand_child_based_get()

class WunderWeather.test.**TestYesterday**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_data_based_get()

test_shorthand_based_get()

test_shorthand_child_based_get()

WunderWeather.test.**main**()

WunderWeather.test.**run_snippet**()

WunderWeather.test.**run_test**()

WunderWeather.test_responses module

WunderWeather.today module

class WunderWeather.today.**Historical**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one (today) almanac type data feature response.

high_avg_temp_c

high_avg_temp_f

low_avg_temp_c

low_avg_temp_f

class WunderWeather.today.**Now**(*data*, **args*, ***kwargs*)
Bases: *WunderWeather.weather_base.WeatherBase*

Wrapper for one (today) conditions type data feature response.

temp_c

temp_f

temp_pretty

weather

WunderWeather.weather module

class WunderWeather.weather.**Extract**(*api_key*, *settings=None*)
Bases: object

Encapsulate logic for extracting weather data

This is the main point of entry to extract data from the weather underground service utilizing their public API.

Notes

Wunderground Doc

URL Request Format: *http://api.wunderground.com/api/<API_KEY>/features/settings/q/query.format*

BASE_URL
str – Base string used for URL generation

FEATURE_URL
str – string template to generate URL for a feature request

HURRICANE_URL
str – string template to generate URL for a hurricane feature request

FEATURE_URL_MAP
dict – Mapping of module's feature key to wunderground's key in the URL

FEATURE_RESPONSE_MAP
dict – Mapping of module's feature key to wunderground's key in the response

FEATURE_CLASS_MAP
dict – Mapping of module's feature key to the object definition to generate an instance

BASE_URL = 'http://api.wunderground.com/api'

FEATURE_CLASS_MAP = {'geolookup': 'weather_base.WeatherBase', 'satellite': 'weather_

FEATURE_RESPONSE_MAP = {'geolookup': 'location', 'satellite': 'satellite', 'weekcast

FEATURE_URL = 'http://api.wunderground.com/api/{key}/{features}/{settings}/q/{query}.{

FEATURE_URL_MAP = {'geolookup': 'geolookup', 'satellite': 'satellite', 'weekcast':

HURRICANE_URL = 'http://api.wunderground.com/api/{key}/{feature}/{settings}/view.{forma

alerts(*query*)
Shorthand to interface with the alerts data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

astronomy(*query*)
Shorthand to interface with the astronomy data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

cams(*query*)
Shorthand to interface with the webcams data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

date(*query*, *date*)
Shorthand to interface with the history data feature

Parameters

- **query** (*str or list*) – string or list of strings for query portion of URL generation
- **date** (*str*) – Date in the form YYYYMMDD.

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns date.Date instance or None

date_range (*query*, *date_range*)
Shorthand to interface with the planner data feature

Parameters

- **query** (*str or list*) – string or list of strings for query portion of URL generation
- **date_range** (*str*) – Date range (30 day max) in the form MMDDMMDD.

feature_context
tuple – tuple of tuples to give feature of interest

and necessary data for that feature

Returns date.Range instance or None

daycast (*query*)
Shorthand to interface with the forcast data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns today.Now instance or None

features (*query*, *feature_context*)
Central logic for making data feature requests

Parameters

- **query** (*str or list*) – string or list of strings for query portion of URL generation
- **feature_context** (*tuple*) – tuple of tuples to supply feature of interest and necessary data for that feature

query
list – Strings for URL generation

feature_codes
str – Final format for URL feature keys (with data appended)

context
dict – final formatted data for URL template

response
dict – JSON response

ctor
obj – Object Class Reference to generate an instance. Could be one of
- `weather_base.WeatherBase`
- `today.Now`
- `today.Historical`
- `forcast.Daycast`
- `forcast.Weekcast`
- `date.Date`
- `date.Range`

weather_features
list – Instances to be retuned. Offset could be None if there was no response for the supplied data feature.

feature_key
str – Module's feature key

response_feature_key
str – Module's feature key for URL response

Returns List of weather_features. Offsets returned in the order they are supplied in the feature_context arg. Offset could be none if no response

geolookup (*query*)
Shorthand to interface with the geolookup data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

classmethod get_feature_class (*feature_key*)
Get the class definition for a particular feature

Parameters **feature_key** (*str*) – internal key for feature

Returns Class definition for feature

hourly_daycast (*query*)
Shorthand to interface with the hourly data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

hourly_weekcast (*query*)
Shorthand to interface with the hourly10day data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather.Weather_Base instance or None

hurricane ()
Interface with the current hurricane data feature

feature_key
str – Module feature key

context
dict – Used to populate feature URL template

response
dict – JSON representation of the response

response_feature_key
str – Module feature key's key for response parsing

ctor
weather.WeatherBase – Reference to class to potentially generate an instance

Returns weather_base.WeatherBase instance or None

rawtide(*query*)
Shorthand to interface with the rawtide data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

satellite(*query*)
Shorthand to interface with the satellite data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

tide(*query*)
Shorthand to interface with the tide data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns weather_base.WeatherBase instance or None

today_historical(*query*)
Shorthand to interface with the almanac data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns today.Historical instance or None

today_now(*query*)
Shorthand to interface with the conditions data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns today.Now instance or None

weekcast(*query*)
Shorthand to interface with the forcast10day data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns forecast.Forecast instance or None

yesterday(*query*)
Shorthand to interface with the yesterday data feature

Parameters **query** (*str or list*) – string or list of strings for query portion of URL generation

feature_context
tuple – tuple of tuples to give feature of interest and necessary data for that feature

Returns date.Date instance or None

WunderWeather.weather_base module

class WunderWeather.weather_base.**WeatherBase**(*data*)
Bases: object

Wrapper for one all data feature responses.

data
EasyDict – dictionary that allows for 'dotted' key references

__data
dict – python dict, JSON representation of portion of response of interest

NAN
list – list of values that are considered to be no data N/A type values

NAN = [-999, -9999]

extract_value(*keys*)
constructor to interface with feature response

Parameters **keys** (*list*) – list of keys to drill down into nested dictionaries

Returns value of interest or None

Module contents

CHAPTER 2

Looking to Contribute

Thanks for checking out this section and showing interest in making this package better. The following are points of interest that could use polishing or expanding. As always, if you see data points across data features that could use a level of abstraction just add a wrapper class if not already defined and add a property member to that class to provide a shortcut or normalized external name across features.

2.1 TODOs

1. **Several Data Features only exist using the generic WeatherBase, base class and thus their data is accessed using the data n**

    (a) currenthurricane

    (b) rawtide and tide

    (c) hourly* based features

2. Of course, help with documentation, documentation, and more documentation.

CHAPTER 3

Mentions

I just want give mention and thanks to the following:

1. Weather Underground for supplying the data.
2. requests for making http for me.
    - requests github
3. EasyDict for supplying the dotted dictionary notation functionality.
    - EasyDict github

Contributors:

Tyler Santos

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

f

t

w

Index

Symbols

A

B

C

D

E

F

G

H

W

Y