PyCrest Documentation

Release 0.0.1

Dreae

Mar 12, 2017

Contents

PyCrest aims to provide a quick and easy way to interact with EVE Online's CREST API

CHAPTER 1

Installation

PyCrest can be installed from PyPi with pip:

```
$ pip install pycrest
```

CHAPTER 2

Getting Started

The entry point for the package should be the pycrest.EVE class

```
>>> import pycrest
>>> eve = pycrest.EVE()
```

The above code will create an instance of the class that can be used for exploring the EVE public CREST data. The connection must be initialized before requests can be made to the CREST API. Loading is done by calling the class instance, and consecutive calls will not produce additional overhead:

```
>>> eve()
```

Attempting to access CREST resources before the connection is loaded will produce an exception.

Resources for the CREST data are mapped as attributes on the EVE class, allowing you to easily traverse them:

```
>>> eve.incursions
{u'href': u'https://crest-tq.eveonline.com/incursions/'}
```

In order to access resources that must be fetched from the API first, you must call the desired resource:

```
>>> eve.incursions
{u'href': u'https://crest-tq.eveonline.com/incursions/'}
>>> eve.incursions()
{u'items': [{...}], u'totalCount_str': u'5', u'totalCount': 5, u'pageCount': 1, u
↪'pageCount_str': u'1'}
>>> eve.incursions().totalCount
5
```

Some useful helper methods to make your life easier / improve readability of next example:

```
>>> def getByAttrVal(objlist, attr, val):
...     ''' Searches list of dicts for a dict with dict[attr] == val '''
...     matches = [getattr(obj, attr) == val for obj in objlist]
...     index = matches.index(True)  # find first match, raise ValueError if not found
...     return objlist[index]
```

```
...
>>> def getAllItems(page):
...     ''' Fetch data from all pages '''
...     ret = page().items
...     while hasattr(page(), 'next'):
...         page = page().next()
...         ret.extend(page().items)
...     return ret
...
```

You can also pass parameters to resources supporting/requiring them, eg. *type* parameter for the regional market data endpoint:

```
>>> region = getByAttrVal(eve.regions().items, 'name', 'Catch')
>>> item = getByAttrVal(getAllItems(eve.itemTypes), 'name', 'Tritanium').href
>>> getAllItems(region().marketSellOrders(type=item))
[{u'price': 9.29, u'volume': 1766874, u'location': {'name': u'V-3YG7 VI - EMMA STONE␣
↪NUMBER ONE', ...}, ...}, ... ]
```

By default resources are cached in-memory, you can change this behaviour by passing the *cache* keyword argument to the EVE class, with an instance of an object that implements APICache. There are several caches available in the pycrest.cache module.

pycrest.cache.DictCache is the default cache and caches requests in memory. Any cached objects will be lost when the process terminates. pycrest.cache.FileCache caches requests on disc. It has one keyword argument named 'path' which configures a directory where file objects will be stored. pycrest.cache.MemcachedCache connects to a memcached server and stores objects in memory. This is very fast and has the advantage over DictCache that the cached objects can be shared across multiple processes. It has one keyword argument named 'server_list' which is a list of memached servers to use (for example ['127.0.0.1:11211']). This cache requires python-memcached to be installed. pycrest.cache.DummyCache doesn't cache anything.

```
>>> import pycrest
>>> from pycrest.cache import FileCache
>>> file_cache = FileCache(path='/tmp/pycrest_cache')
>>> eve = pycrest.EVE(cache=file_cache)
```

CHAPTER 3

Authorized Connections

PyCrest can also be used for accessing CREST resources that require an authorized connection. To do so you must provide the EVE class with a *client_id*, *api_key*, and *redirect_uri* for the OAuth flows for authorizing a client. Once done, PyCrest can be used for obtaining an authorization token (short-lived) and a refresh token (long-lived):

```
>>> eve = pycrest.EVE(client_id="your_client_id", api_key="your_api_key", redirect_
↪uri="https://your.site/crest")
>>> eve.auth_uri(scopes=['publicData'], state="foobar")
'https://login.eveonline.com/oauth/authorize?response_type=code&redirect_uri=...'
```

Once you have redirected the client to acquire authorization, you may pass the returned code to *EVE.authorize()* to create an authorized connection. The code is returned as a parameter of the callback URI that you specified when you registered your application with CCP. For example, if your callback URI is 'https://callback.example.com/callback/' then the code would be returned by redirecting to something like 'https://callback.example.com/callback/?code=a1b2c3djsdfklsdfjklfsdjflk'.

```
>>> eve.authorize(code)
<pycrest.eve.AuthedConnection object at 0x024CD8F0>
```

The authorized API connection functions identically to the public connection, except that requests will be directed to the authorized CREST endpoint. You can retrieve information about the authorized character by calling *whoami()* on an authorized connection:

```
>>> con = eve.authorize(code)
>>> con.whoami()
{u'Scopes': u'publicData', u'CharacterName': u'Dreae', ...}
```

Note that currently CREST authorization tokens expire after 1200 seconds and are automatically refreshed upon expiry. You can also refresh tokens manually by calling *refresh()* on the authorized connection. This refreshes the connection in-place and also returns *self* for backward compatibility.

```
>>> con.refresh()
<pycrest.eve.AuthedConnection object at 0x0251F490>
```

Refresh Tokens

Once the authorization token has expired (perhaps after you restart the application), you can obtain a new authorization token either by having the user go through the log-in process again or by using the long-lived refresh token to get a new authorization token without involving the user. The OAuth2 refresh token is stored inside the AuthedConnection object you obtained earlier, so you should persist this token somewhere safe for later use. Note that the refresh_token should be stored securely, as it allows anyone who possesses it access to whichever scopes you authorized. To get a new authorization token from the refresh token, pass the refresh token to refr_authorize(), and you have an Authed-Connection object ready to access CREST.

```
>>> con = eve.authorize(returnedCode)
>>> con
<pycrest.eve.AuthedConnection object at 0x7f06dd61e410>
>>> con.refresh_token
u'djsdfjklsdf9sd8f908sdf9sd9f8sd9f8sdf8sp9fd89psdf89spdf89spdf89spdf89p'
```

```
>>> eve.refr_authorize(refresh_token)
<pycrest.eve.AuthedConnection object at 0x7f06e21f48d0>
```

Prevent PyCrest from using cache

No cache for everything in PyCrest This will disable the cache for everything you will do using PyCrest. No call or response will be stored.

```
>>> pycrest_no_cache = EVE(cache=None)
```

Disable caching on demand You can disable the caching for a specific `get()` call you don't want to cache, by simply adding `caching=False|None` to the call parameters. For example:

```
>>> crest_with_caching = EVE()
>>> crest_root_not_cached = crest_with_caching(caching=False)
>>> regions = crest_root.regions(caching=False)
```