pymagento Documentation

Release 0.1

Russ Amos

Apr 17, 2017

Contents

pymagento provides Python bindings for the Magento Core API.

CHAPTER 1

Installation

```
pip install pymagento
```

CHAPTER 2

Usage

```
import pymagento
api = pymagento.Magento("hostname", "api_user", "api_key")
category_id = api.category.create(1, {"name": "New Category"})
category_info = api.category.info(category_id)
arbitrary_product = api.product.list()[39]
api.category.assignProduct(arbitrary_product["id"])
```

CHAPTER 3

Contents

Filtering `catalog_product` queries

This document is not a substitute for the real Magento API documentation. Much of this information was taken from Using Collections in Magento .

`api.catalog_product.list`

`api.catalog_product.`**`list`**`(`[*filters : dict*[, *storeView : int*]]`)` → list of dicts

Retrieve a product list

Produce an `AND` query by adding multiple items to the filter dict:

```
{
    'sku': {'null': 'true'},
    'product_id': {'eq': '3047'},
}
```

Produce an `OR` query by passing a list of filter dicts:

```
{
    'product_id': [
        {'eq': '3047'},
        {'eq': '2979'},
    ],
}
```

Available filters:

```
like
nlike:

    {'sku': {'like': 'S8MST-E13W-WHT-%'}}
```

```
eq
neq:

    {'sku': {'neq': ''}}

notnull
null:

    {'sku': {'null': 'true'}}

in
nin:

    {'sku': {'in': ['S8MST-E13W-WHT-SM', 'S8MST-E13W-WHT-MD']}}

is
gt
lt
gteq
moreq
lteq:

    {'status': {'lteq': '1'}}

finset (mysql's FIND_IN_SET()):

    ??

from -> to (uses dates)
date -> to (converts to date)
datetime -> to (converts to datetime):

    {'created_at': {'from': '2011-09-13 15:31:21'}}

    # converts comparison value types
    {'created_at': {'from': '10 September 2000',
            'to': '11 September 2000',
            'date': 'true'}}
```

Example of using `AND` and `OR` queries together:

```
>>> STORE_ID = 1
>>> all_prods = api.catalog_product.list({
...     'status': {'eq': '1'},
...     'sku': [{'neq': ''}, {'notnull': 'true'}],
...     'upc': [{'neq': ''}, {'notnull': 'true'}],
... }, STORE_ID)
```

cataloginventory_stock_item.list

api.cataloginventory_stock_item.**list**(*[products : list]*) → list of dicts

Retrieve stock data by product ids

Usage:

```
api.product_stock.list(['S8MST-E14W-WHT-MD', 'S8MST-E14W-WHT-SM'])
```

Magento's `multiCall` method

Magento's `multiCall` method (not to be confused with the `xmlrpclib` function of the same name) allows multiple API methods to be called in one request which can drastically speed up multiple queries.

The `multiCall` wrapper breaks from the normal `pymagento` usage to make it easier to programatically generate bulk queries. For example:

```
>>> all_prods = api.catalog_product.list({
...     'sku': [{'neq': ''}, {'notnull': 'true'}],
... })
>>> len(all_prods)
200
>>> all_skus = [i['sku'] for i in all_prods]
>>> all_prod_details = api.multiCall([
...     [['catalog_product.info', [sku]] for sku in all_skus],
...     [['catalog_product_attribute_media.list', [sku]] for sku in all_skus],
... ])
```

Note: `multiCall` does not raise faultCode exceptions

With normal `pymagento` use, if you call a server method incorrectly (by calling a method that does not exist or calling a method with incorrect parameters) `xmlrpclib` will raise a `Fault` exception:

```
>>> api.thisdoesnotexist.info()
Traceback (most recent call last):
    ...
Fault: <Fault 3: 'Invalid api path.'>
```

When using Magento's `multiCall` method those `Fault` codes are embeded in the return data structure so you will have to look for them manually:

```
>>> api.multiCall([['thisdoesnotexist', []]])
[{'faultCode': '3', 'faultMessage': 'Invalid api path.', 'isFault': True}]
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

A