Python CiviCRM Documentation

Release 0.1

Paul Munday

March 04, 2015

Contents

:synopis:Python module to access the CiviCRM v3 API.

CHAPTER 1

Python CiviCRM

This is a module for interacting with the CiviCRM REST API Version 3.

It's use should be fairly straight forward, at least if you have basic familiarity with the API documentation.

Everything is implemented in the CiviCRM class. You need, at a minimum to pass a URL, and the your site and API keys when instantiating your object.

This class has methods that correspond to nearly all the underlying actions provided by the REST API, as well as implementing a few more of its own. Some of these are more convenient and/or simpler versions of the actions. Others provide for more complex or specific tasks e.g. adding a contact. The eventual aim is to match the functionality provided in the PHP API.

There are some differences in how things are implemented, in particular with how things are returned. See *Things to note*.

A CivicrmError will be raised for anything other than a 200 respose e.g. a 404.

1.1 Usage

Usage example for a basic search:

```
url = 'www.example.org/path/to/civi/codebase/civicrm/extern/rest.php'
site_key ='your site key'
api_key ='your api key'
civicrm = CiviCRM(url, site_key, api_key)

search_results = civicrm.get('Contact', city='Gotham City')
first_10_search_results = civicrm.get('Contact',
        city='Gotham City', limit=10)
```

It can be easier to construct a dict and feed them to methods using ** to expand it to key value pairs:

```
my_dict =   {
            country      = 'United States',
            city         ='Gotham City',
            contact_type ='Individual
            }
civicrm.get('Contact', **my_dict)
```

The following optional values can be supplied when intializing: use_ssl=True/False Connect over https not http, defaults to True. timeout=N Connection will time out in N seconds, i.e if

no response is sent by the server in N seconds. requests.exceptions.Timeout will be raised if the connection timesout. Defaults to None, this means the connection will hang until closed.

e.g. url = 'www.example.org/path/to/civi/codebase/civicrm/extern/rest.php' site_key ='your site key' api_key ='your api key' civicrm = CiviCRM(url, site_key, api_key, timeout=5)

Connections will timeout after 5 seconds, and raise an error.

1.2 Things to note

- Of the options defined in the CiviCRM API

http://wiki.civicrm.org/confluence/display/CRMDOC/Using+ the+API#UsingtheAPI- Parameters only limit, offset (& sequential) are currently supported, sequential is set to 1 (true) by default and should not generally be changed.

- Entity and Action must always be specified explicitly. They are removed if

found in params, along with references to site/api keys.

- The CiviCRM API typically returns JSON (that would decode to a dictionary)

with the actual results you want stored in values(or result if a single value is expected). Additional info is typically API version and count. If results are returned successfully we only return the results themselves – typically a list of dictionaries, as this API version is always 3 with this module, and count can easily be derived using len().

- Returned values are generally sequential (i.e. a list (of dictionaries)

rather than a dictionary (of dictionaries) with numbers for keys) except in the case of getfields & getoptions that return a dictionary with real keys.

- Results are unicode
- Most actions returns the (updated) record in question, others a count e.g.
- delete
- The is_valid_method uses getoptions() to both checks an option is valid and

returns the corresponding id (where this is valid). It does this for both an id supplied as an int and a label supplied as a string. This is convinient as it allows methods to take a descriptive label as well as a numeric id, rather than being limited to this. When methods do this they don't use this method if a numeric id is supplied, so that id is not checked for validity (though a Civicrm Error will still be raised if its not) as it is assumed you know what you are doing in this case, and we can save an extra API call for speed.

- The replace API call is undocumented, AFAIK, so not implemented, use

getaction if you must.

class pythoncivicrm.**CiviCRM**(*url*, *site_key*, *api_key*, *use_ssl=True*, *timeout=None*)

Make calls against the Civicrm API.

add_activity(*activity_type*, *sourceid*, *subject=None*, *date_time=None*, *activity_status=None*, *activity_medium=None*, *priority=None*, ***kwargs*)

Creates an activity. activity_type, activity_status, activity_medium and priority can all be supplied as a label or id (int). The label wil be automatically coverted into the corresponding id so activity_type becomes activity_type_id. Valid values can be obtained with the getoptions method e.g. getoptions('Activity', 'status_id') (This method doesn't take activity_type_name – its identical for predefined types. use the create method if you insist on using it). sourceid is an int, typically the contact_id for the person creating the activity, loosely defined. There is also a target_contact_id for person contacted etc. Subject is a string, typically a summary of the activity. date_time should be string not a datetime object. It's short hand for 'activity_date_time'.

add_activity_type(*label*, *weight=5*, *is_active=0*, ***kwargs*)
Creates an Activity Type. Label is a string describing the activity spaces are allowed. Weight is any postive or negative integer. It affects the order in which things are displayed in the web interface. It defaults to 5, this puts things just after the basic types such as Phone Call. is_active defaults to 0: disabled (as per CiviCRM. Set to 1 to make the Activity Type active.

add_address(*contact_id*, *location_type*, ***kwargs*)
Add an address to civicrm. location_type can be supplied as numeric id or its equivalent value.

add_contact(*contact_type*, ***kwargs*)
Creates a contact from supplied dictionary params. Raises a CivicrmError if a required field is not supplied: contact_type and/or one of first_name, last_name, email, display_name. Returns a dictionary of the contact created.

add_contribution(*contact_id*, *total_amount*, *financial_type*, ***kwargs*)
Add a contribution of amount credited to contact_id. financial_type can be an integer or a string corresponding to a financial types id or value respectively. This can be obtained with self.getoptions('Contribution', 'financial_type_id').

add_email(*contact_id*, *email*, *email_like=False*, ***kwargs*)
Add an email to civicrm. If email_like is True it checks to see whether the supplied email looks something like a real email, using a typical handwavey regex (specifically something like something@something.something so local emails will fail). A CivicrmError is raised if it fails this "test". No claim is made that this actually is or isn't a valid email, never mind that you can actually send email to it. Civicrm doesn't care and will take anything in the field apparently.

add_entity_tag(*entity_id*, *tag_id*, *entity_table=u'civicrm_contact'*)
Tag an entity_id (a contact id by default) by tag id. Note returns a dict with "is_error,not_added, added, total_count It's not an error to tag an entity with a tag, it just won't get added Iand added and not_added will reflect this. See also notes under delete.

add_group(*title*, ***kwargs*)
Add a group to CiviCRM.

add_group_contact(*contact_id*, *group_id*, ***kwargs*)
Add a link between a group and a contact. See entity_tag for a description of return values (and deleting).

add_note(*entity_id*, *note*, ***kwargs*)
Add a note . Note if entity_table is not defined, it defaults to civicrm_contact. entity_table refers to the table name in civicrm database. Other fields are subject contact_id (note creator), modified_date and privacy.

add_phone(*contact_id*, *phone*, ***kwargs*)
Add a phone number to CiviCRM. phone_type is an int, is_primary defaults to 1(true). phone_numeric is phone number as digits only (no spaces, dashes etc).

add_relationship(*contact_a*, *contact_b*, *relationship*, ***kwargs*)
Adds a relationship between contact_a and contact_b. Contacts must be supplied as id's (int). If the relationship is supplied as an int it is assumend to be an id, otherwise name_a_b, label_a_b, name_b_a, label_b_a and description are searched for a match. A CivicrmError is raised if no match is found. N.B. 'Alice', 'Bob', 'Employer of' means Bob is the employer of Alice. Non compulsory fields may be passed in a keyword pairs. Searching for a match will hit the API and may do so multiple times, you may find it beneficial to check the result for 'relationship_type_id' and cache this result. Returns a dictionary of the contact created.

add_tag(*name*, ***kwargs*)
Add a tag.

create(*entity*, ***kwargs*)
Simple implementation of create action. Returns a list of dictionaries of created entries.

delete(*entity*, *db_id*, *skip_undelete=False*)
Delete a record. Set skip_undelete to True, to permanently delete a record for cases where there is a 'recycle bin' e.g. contacts. In some cases, e.g. entity_tags entries entries can be deleted without the id (get on entity tags doesn't return the id). In these cases use getaction with delete and the appropriate key-value pairs. Returns the number of deleted records.

doaction(*action*, *entity*, ***kwargs*)
There are other actions for some entities, but these are undocumented?. This allows you to utilise these. Use with caution.

get(*entity*, ***kwargs*)
Simple implementation of get action. Supply search terms in a dictionary called params Pass limit and offset for a subset of the results (or pass them in params). Other options can also be passed as key=value pairs (options as defined here: http://wiki.civicrm.org/confluence/display/CRMDOC/Using+the+API #UsingtheAPI-Parameters e.g. match, match mandatory. Returns a list of dictionaries or an empty list.

getcount(*entity*, ***kwargs*)
Returns the number of qualifying records. Expects a dictionary. May not be accurate for values > 25. (will return 25).

getfields(*entity*)
Returns a dictionary of fields for entity, where keys (and key['name']) are names of field and the value is a dictionary describing that field.

getoptions(*entity*, *field*)
Returns a dictionary of options for fields as key/value pairs. Typically identical to each other. (though sometimes appear to be synonyms? e.g. 1: Yes) Raises CivicrmError if a field has no associated options or is not present etc.

getsingle(*entity*, ***kwargs*)
Simple implementation of getsingle action. Returns a dictionary. Raises a CiviCRM error if no or multiple results are found.

getvalue(*entity*, *returnfield*, ***kwargs*)
Simple implementation of getvalue action. Will only return one field as unicodestring and expects only one result, as per get single. Raises a CiviCRM error if no or multiple results are found.

is_valid_option(*entity*, *field*, *value*)
Takes a value which can be an id or its corresponding label, Returns the (corresponding) id if valid, otherwise raises a CivicrmError.

setvalue(*entity*, *db_id*, *field*, *value*)
Updates a single field. This is not well documented, use at own risk. Takes an id and single field and value, returns a dictionary with the updated field and record.

update(*entity*, *db_id*, ***kwargs*)
Update a record. An id must be supplied. Returns a list of dictionaries of updated entries.

pythoncivicrm.**matches_required**(*required*, *params*)
if none of the fields in the list required are in params, returns a list of missing fields, or None

p

A

C

D

G

I

M

P

S

U