Archived:A guided tutorial using Twitter API

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee
(27 Jul 2011)
The article is accurate but refers largely to now-deprecated Twitter api methods. See https://dev.twitter.com/docs/api for current list of methods.

Introduction

Several recent Web Services like Twitter, Flickr and Yahoo! Maps have been presented with REST as one access option. The Representational State Transfer (REST), as named by its creator, Roy Fielding, is defined as a architectural style for distributed hypermedia systems. More than people usually call "web service", *REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems* (Roy Fielding).

Some benefits are usually credited to REST services, like:

- Cache support, decreasing response time and requiring less server load.
- Sessions are not necessary, allowing a better load distribution among several servers.
- Client-server architecture, with few requirements for the client-side. In general only a browser is necessary.
- Better long-term compatibility

Due to intrinsic "web nature" of REST, a suitable Python module for accessing REST services is urllib. Some urllib features as described in next section.

urllib module

urllib is a versatile python module for fetching data across the Internet. It has several interesting features, like:

- opening URLs with an interface similar to that one found in file operations.
- function for processing URL, like escaping HTML and parameter processing
- proxy and HTTP basic authentication support

Powerful programs can be created with urllib. For instance, suppose you want to fetch the content of page "http://www.developer.nokia.comhttp://developer.nokia.com/Community/Wiki/Wiki_Home" and save it into local file **"wikiforumnokia.html"**. This can be performed by urllib using few lines of code:

urllib_demo1.py

```
import urllib

# returns a file like interface
furl = urllib.urlopen("http://www.developer.nokia.comhttp://developer.nokia.com/Community/Wiki/Wiki_Home")
# reading the "file"
contents = furl.read()
# saving the page contents
flocal = open("wikiforumnokia.html","wt")
flocal.write(contents)
```

Or, if you prefer, **urlretrieve()** method can make this job with just two lines of code:

urllib_demo2.py

```
import urllib
urllib.urlretrieve("http://www.developer.nokia.comhttp://developer.nokia.com/Community/Wiki/Wiki_Home","wikiforumnokia.html")
```

In fact, **urlopen()** performs an HTTP GET request and fetches the contents of the desired page, striping the HTTP header. If additional parameters are necessary in your request, they may be added to the URL, like a typical URL GET request.

urllib_demo3.py

```
import urllib
params = urllib.urlencode({'name': 'My name is bond', 'phone':'007007'})
url = "www.exemplo.com/yourname?" + params
print url
```

The output is an URL with all parameters encoded, as you can see in the address bar when searching at Yahoo! or Google.

(C) Copyright Nokia 2013. All rights reserved.

```
>>> 'www.exemplo.com/yourname?phone=007007&name=My+name+is+bond'
```

It is possible to simulate an HTTP POST request as well. POST requests do not use the URL to encode the parameters. Instead, parameters are included in the body of the request. Forms are a good example of POST requests, where all form parameters are hidden inside html body.

We can access **www.exemplo.com** with same previous arguments but now using POST, as demonstrated in the following code snippet.

urllib_demo4.py

```
import urllib

params = urllib.urlencode({'name': 'My name is bond', 'phone':'007007'})
result = urllib.urlopen("www.exemplo.com/yourname",params).read()
```

In this case, an additional parameter must be supplied to **urlopen**, indicating the POST request.

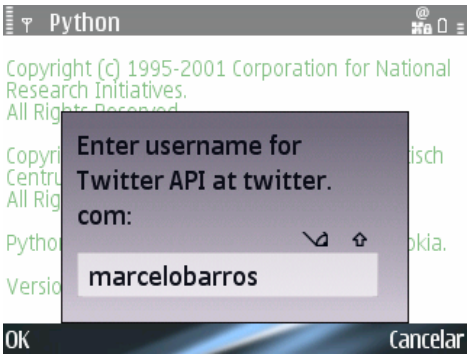
POST requests may be translated as **create**/**delete** operations in REST and GET as **read** operations. They are basic methods for accessing and modifying your remote REST resources, as it will be demonstrated in this article.

One more awesome urllib feature is HTTP basic authentication. With basic authentication is possible to specify your credentials (user and password) when accessing some URL. urllib handles authentication requests, given to the user the chance to type your username and password. This feature is specially useful for Twitter API and it is demonstrated in next example (the Twitter URL used will be explained later, don't worry).

urllib_demo5.py

```
import urllib

url = 'http://twitter.com/statuses/friends_timeline.json'
f = urllib.urlopen(url)
d = f.read()
```

S60 devices will ask the user for username and password, as in pictures below:



urllib asking username

urllib asking password

However, these dialogs can be avoided using FancyURLopener, an urllib class for handling some special HTTP response codes, like 401 (authentication required). In this case, it is necessary to overload the method **prompt_user_passwd**, using it to provide the tuple (username,password) when requested by HTTP server, as showed in next example.

urllib_demo6.py

```
import urllib

class _FancyURLopener(urllib.FancyURLopener):
    """ This class handles basic auth, providing user and password
        when required by http response code 401
    """
    def __init__(self, usr, pwd):
        """ Set user/password for http and call base class constructor
        """
        urllib.FancyURLopener.__init__(self)
        self.usr = usr
        self.pwd = pwd
```


(C) Copyright Nokia 2013. All rights reserved.

```
    def prompt_user_passwd(self, host, realm):
        """ Basic auth callback
        """
        return (self.usr,self.pwd)

# Create a customized urlopener for handling authentication request
urlopener = _FancyURLopener("twitter_username","twitter_password")
# read friends timeline, using credential provided
f = urlopener.open('http://twitter.com/statuses/friends_timeline.json')
# twitter response
d = f.read()
```

POST/GET requests and basic HTTP authentication are two important elements when communicating to Twitter API. However, for a complete communication with Twitter, one essential component is still missing: object serialization.

Object serialization

Twitter API can send responses using XML, JSON, RSS or Atom. The response format depends on request URL. For instance, when requesting the user friends timeline, the following URL is used:

```
http://twitter.com/statuses/friends_timeline.format
```

In this case, caller must choose the desired response format, using json, xml, rss or atom as **format**. For JSON responses, the timeline request must be:

```
http://twitter.com/statuses/friends_timeline.json
```

JavaScript Object Notation (JSON) is a lightweight data-interchange format, easily readable for humans and equally easily parsed and generated by machines. Several languages, including Python, have one (or several) JSON implementation. Talking about PySymbian, at least three implementations are available:

- json.py, by Patrick D. Logan and Ville H. Tuulos.
- simplejson 2.0.9 in single file, ported by Aapo Rista (inspired by Marcelo Barros).
- simplejson, ported by Marcelo Barros, with just few modifications when compared to the original implementation (this article will use this implementation).

Using JSON, almost all basic types may be encoded as strings and sent over an Internet connection. Two simplejson methods (**dumps** and **loads**) are responsible for encoding data as a JSON representation and decoding it afterwards. This way, information sent by Twitter may be received and properly processed.

For instance, consider the following dictionary and its posterior serialization/unserialization:

json_demo.py

```
import simplejson

d = { "name":"Marcelo", "age": 36, "weight":1.77, "devices":["N95", "E71", "N800"] }
ser = simplejson.dumps(d)

print len(ser)
print type(ser)
print ser

rd = simplejson.loads(ser)

print len(rd)
print type(rd)
print rd
```

The serialization output is a string, ready to be transmitted over some TCP/IP connection:

```
>>> 81
>>> <type 'str'>
>>>  '{"age": 36, "devices": ["N95", "E71", "N800"], "name": "Marcelo", "weight": 1.77}'
```

This string can be converted to a Python object again using the **loads** method, as showed in the following output:

```
>>> 4
>>> <type 'dict'>
>>> {u'age': 36, u'weight': 1.77, u'name': u'Marcelo', u'devices': [u'N95', u'E71', u'N800']}
```

Object serialization is the last element for understanding Twitter API. Next section will present a base class for using Twitter API.

Twitter API

Twitter REST API is well documented and it is a must read for developers building tools that talk to Twitter. Although this tutorial will cover only three methods, they are enough for a clear understanding of Twitter API. The covered methods are:

Warning: Some of the methods below are deprecated. This article needs to be updated.

- get_friendstimeline: Returns the 20 most recent statuses posted by the authenticating user and that user's friends.

(C) Copyright Nokia 2013. All rights reserved.

- update: Add an update to the authenticating user's statuses.
- destroy: Destroys a status.

The code is below. Some parts were already explained in previous sections, like HTTP authentication, JSON and urllib usage.

s60twitter.py

```
import urllib
import simplejson as json

class _FancyURLopener(urllib.FancyURLopener):
    """ This class handles basic auth, providing user and password
        when required by http response code 401
    """
    def __init__(self, usr, pwd):
        """ Set user/password for http and call base class constructor
        """
        urllib.FancyURLopener.__init__(self)
        self.usr = usr
        self.pwd = pwd

    def prompt_user_passwd(self, host, realm):
        """ Basic auth callback
        """
        return (self.usr,self.pwd)

class TwitterApi(object):
    """ Twitter API basic class
    """
    def __init__(self, tw_usr, tw_pwd):
        """ Set user/password for twitter
        """
        self._tw_usr, self._tw_pwd = tw_usr, tw_pwd

    def _get_urlopener(self):
        """ Return an urlopener with authentication support
        """
        return _FancyURLopener(self._tw_usr, self._tw_pwd)

    def get_friends_timeline(self, page=1, count=20):
        """ Return friends timeline for current user
        """
        params = urllib.urlencode({"page":page,"count":count})
        url = "http://twitter.com/statuses/friends_timeline.json?" + params
        f = self._get_urlopener().open(url)
        d = f.read()
        return json.loads(d)

    def update(self, stat_msg):
        """ Update twitter with new status message
        """
        params = urllib.urlencode({"status":stat_msg})
        url = "http://twitter.com/statuses/update.json"
        f = self._get_urlopener().open(url, params)
        d = f.read()
        return json.loads(d)

    def destroy(self,udpt_id):
        """ Destroy the status specified by udpt_id
        """
        url = "http://twitter.com/statuses/destroy/%s.json" % udpt_id
        f = self._get_urlopener().open(url,"")
        d = f.read()
        return json.loads(d)
```

Each method is represented as an URL with the encoding method (json)at the end and some additional parameters. For **get_friends_timeline**, parameters are added to the URL, creating an authenticated GET request to Twitter server. The page number may be specified starting from 1, that is, the most recent page with at most "count" Twitter updates. The response is decoded using simplejson and a Python array of updates is returned to the caller. Each entry in this array is a dictionary with keys representing several update fields. In special:

- **text**: update text
- **created_at**: update date
- **user**: information about author
- **id**: update ID

It is possible to send an update to Twitter using **update** method. In this case, an authenticated POST request is used and the update text is encoded in html body.

Finally, an update may be deleted with method **destroy**. It is an special URL where the update ID is inserted at the end. Again, a authenticated POST request is used, even with no parameters.

More methods can be implemented using the last ones as example. Just check Twitter RESP API.

Application example

A small application that uses TwitterApi is below. It uses the Archived:PySymbian basic user interface app framework. Just put your twitter username (**self.twitter_user**) and password (**self.twitter_password**), copy the files to memory card (**e:\python**) and run it. Use left/right navigation keys for moving in pages and menu option for refresh, send an update or delete an update.

twitter_demo.py

```
# -*- coding: utf-8 -*-
import sys
sys.path.append(r"e:\python")
import key_codes
from appuifw import *
```


(C) Copyright Nokia 2013. All rights reserved.

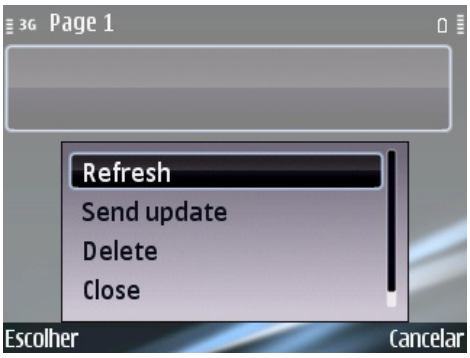
```
from window import Application, Dialog
from s60twitter import TwitterApi

__all__ = [ "twitter_demo" ]
__author__ = "Marcelo Barros de Almeida (marcelobarrosalmeida@gmail.com)"
__version__ = "0.0.1"
__copyright__ = "Copyright (c) 2009- Marcelo Barros de Almeida"
__license__ = "GPLv3"

class Notepad(Dialog):
    """ Minimum text editor for writing new updates
    """
    def __init__(self, cbk, txt=u""):
        menu = [(u"Send", self.close_app),
                (u"Discard", self.cancel_app)]
        Dialog.__init__(self, cbk, u"New update", Text(txt), menu)

class TwitterDemo(Application):
    """ Twitter API demo programa
    """
    def __init__(self):
        menu = [(u"Refresh", self.refresh_pages),
                (u"Send update", self.send_update),
                (u"Delete",self.delete),
                (u"Close", self.close_app)]
        self.body = Listbox([(u"", u"")])
        Application.__init__(self,  u"What I am doing ...", self.body, menu)
        self.dlg = None
        # current displayed page
        self.page = 1
        self.headlines = [(u"",u"")]
        # timeline, indexed by page number
        self.timeline = {}
        self.last_idx = 0
        self.update_msg = ""
        # Twitter credential here !
        self.twitter_user = "twitter_username"
        self.twitter_password = "twitter_password"
        self.twitter_api = TwitterApi(self.twitter_user,self.twitter_password)
        # Some key binding for better navigation
        self.bind(key_codes.EKeyRightArrow, self.inc_page)
        self.bind(key_codes.EKeyLeftArrow, self.dec_page)
        self.bind(key_codes.EKeyUpArrow, self.key_up)
        self.bind(key_codes.EKeyDownArrow, self.key_down)

    def inc_page(self):
        """ Download an older page
        """
        if not self.ui_is_locked():
            self.page += 1
            self.refresh_timeline()

    def dec_page(self):
        """ Download a newer page
        """
        if not self.ui_is_locked():
            self.page -= 1
            if self.page < 1:
                self.page = 1
            else:
                self.refresh_timeline()

    def key_up(self):
        """ Update title with current position
        """
        if not self.ui_is_locked():
            p = app.body.current() - 1
            m = len( self.headlines )
            if p < 0:
                p = m - 1
            self.set_title(u"[%d/%d] Page %d" % (p+1,m,self.page))

    def key_down(self):
        """ Update title with current position
        """
        if not self.ui_is_locked():
            p = app.body.current() + 1
            m = len( self.headlines )
            if p >= m:
                p = 0
            self.set_title(u"[%d/%d] Page %d" % (p+1,m,self.page))

    def refresh_pages(self):
        """ Clear all displayed updates and refresh with page again
        """
        self.timeline = {}
        self.page = 1
        self.refresh_timeline()

    def refresh_timeline(self):
        """ Update timeline info, download current page
        """
        if self.page not in self.timeline:
            self.lock_ui(u"Downloading page %d..." % self.page)
            try:
                self.timeline[self.page] = self.twitter_api.get_friends_timeline(self.page)
            except:
                note(u"Impossible to download page %d..." % self.page,"error")
                self.unlock_ui()
                self.refresh()
                return
            self.unlock_ui()

        self.headlines = []
        # First line: author infor and reply info
        # Second line: update text
        for msg in self.timeline[self.page]:
            r1 = msg[u'user'][u'screen_name']
            if msg[u'in_reply_to_screen_name']:
                r1 += u" to %s" % msg[u'in_reply_to_screen_name']
            r2 = msg[u'text']
            self.headlines.append((r1,r2))
        self.last_idx = 0
        self.refresh()

    def send_update_cbk(self):
        """ Send a new update (dialog callback)
        """
        if not self.dlg.cancel:
            msg = self.dlg.body.get()
```


(C) Copyright Nokia 2013. All rights reserved.

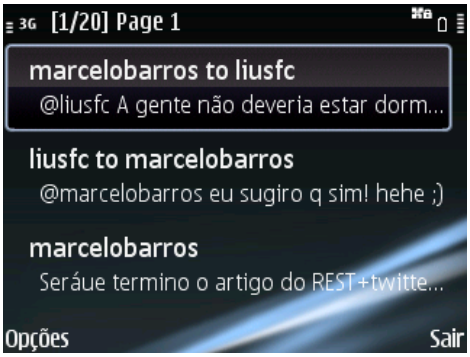
```
            self.set_title(u"Sending update...")
            try:
                # Twitter uses UTF-8
                self.twitter_api.update(msg.encode('utf-8'))
            except:
                note(u"Impossible to send messages. Try again", "error")
                self.unlock_ui()
                return False
        self.unlock_ui()
        self.refresh()
        return True

    def send_update(self):
        """ Send a new update
        """
        self.dlg = Notepad(self.send_update_cbk, self.update_msg)
        self.dlg.run()

    def delete(self):
        """ Delete an update
        """
        idx = self.body.current()
        try:
            updt_id = self.timeline[self.page][idx][u'id']
        except:
            return
        self.lock_ui(u"Deleting ...")
        try:
            self.twitter_api.destroy(updt_id)
        except:
            note(u"Impossible to delete update. Try again", "error")
        else:
            del self.timeline[self.page][idx]
            del self.headlines[idx]
        self.unlock_ui()
        self.refresh()

    def refresh(self):
        """ Refresh Listbox with current statuses
        """
        Application.refresh(self)
        idx = self.body.current()
        if not self.headlines:
            self.headlines = [(u"", u"")]
        self.last_idx = min( self.last_idx, len(self.headlines)-1 )
        app.body.set_list(self.headlines, self.last_idx)
        self.set_title(u"Page %d" % (self.page))

if __name__ == "__main__":

    imd = TwitterDemo()
    imd.run()
```

Screenshots

Some screenshots of this demo running:



Initial menu, with refresh option highlighted. This option will call get_friends_timeline().

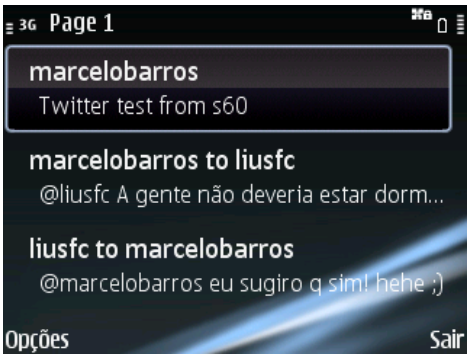


Twitter timeline.

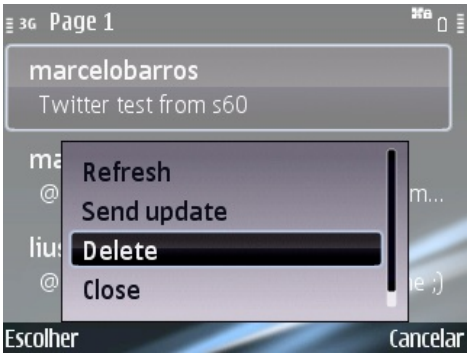
(C) Copyright Nokia 2013. All rights reserved.

Update menu.

Creating and sending a new update.



New update in timeline (use refresh first).



Deleting an update.

Source Code

Donwload source code of all examples in this article: MBA twitter demo src.zip

References

Portuguese version: Trazendo a arquitetura REST para dispositivos PySymbian: um tutorial guiado utilizando Twitter API

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.