Chapter 2
Crawling Twitter Data

Users on Twitter generate over 400 million Tweets everyday.[1] Some of these Tweets are available to researchers and practitioners through public APIs at no cost. In this chapter we will learn how to extract the following types of information from Twitter:

- Information about a user,
- A user's network consisting of his connections,
- Tweets published by a user, and
- Search results on Twitter.

APIs to access Twitter data can be classified into two types based on their design and access method:

- REST APIs are based on the REST architecture[2] now popularly used for designing web APIs. These APIs use the pull strategy for data retrieval. To collect information a user must explicitly request it.
- Streaming APIs provides a continuous stream of public information from Twitter. These APIs use the push strategy for data retrieval. Once a request for information is made, the Streaming APIs provide a continuous stream of updates with no further input from the user.

They have different capabilities and limitations with respect to what and how much information can be retrieved. The Streaming API has three types of end-points:

- Public streams: These are streams containing the public Tweets on Twitter.
- User streams: These are single-user streams, with to all the Tweets of a user.
- Site streams: These are multi-user streams and intended for applications which access Tweets from multiple users.

[1] http://articles.washingtonpost.com/2013-03-21/business/37889387_1_tweets-jack-dorsey-twitter

[2] http://en.wikipedia.org/wiki/Representational_state_transfer

S. Kumar et al., *Twitter Data Analytics*, SpringerBriefs in Computer Science,
DOI 10.1007/978-1-4614-9372-3_2, © The Author(s) 2014

As the Public streams API is the most versatile Streaming API, we will use it in all the examples pertaining to Streaming API.

In this chapter, we illustrate how the aforementioned types of information can be collected using both forms of Twitter API. Requests to the APIs contain parameters which can include hashtags, keywords, geographic regions, and Twitter user IDs. We will explain the use of parameters in greater detail in the context of specific APIs later in the chapter. Responses from Twitter APIs is in JavaScript Object Notation (JSON) format.[3] JSON is a popular format that is widely used as an object notation on the web.

Twitter APIs can be accessed only via authenticated requests. Twitter uses *Open Authentication* and each request must be signed with valid Twitter user credentials. Access to Twitter APIs is also limited to a specific number of requests within a time window called the *rate limit*. These limits are applied both at individual user level as well as at the application level. A *rate limit window* is used to renew the quota of permitted API calls periodically. The size of this window is currently 15 min.

We begin our discussion with a brief introduction to OAuth.

2.1 Introduction to Open Authentication (OAuth)

Open Authentication (OAuth) is an open standard for authentication, adopted by Twitter to provide access to protected information. Passwords are highly vulnerable to theft and OAuth provides a safer alternative to traditional authentication approaches using a three-way handshake. It also improves the confidence of the user in the application as the user's password for his Twitter account is never shared with third-party applications.

The authentication of API requests on Twitter is carried out using OAuth. Figure 2.1 summarizes the steps involved in using OAuth to access Twitter API. Twitter APIs can only be accessed by applications. Below we detail the steps for making an API call from a Twitter application using OAuth:

1. Applications are also known as consumers and all applications are required to register themselves with Twitter.[4] Through this process the application is issued a consumer key and secret which the application must use to authenticate itself to Twitter.
2. The application uses the consumer key and secret to create a unique Twitter link to which a user is directed for authentication. The user authorizes the application by authenticating himself to Twitter. Twitter verifies the user's identity and issues a OAuth verifier also called a PIN.

[3] http://en.wikipedia.org/wiki/JSON

[4] Create your own application at http://dev.twitter.com

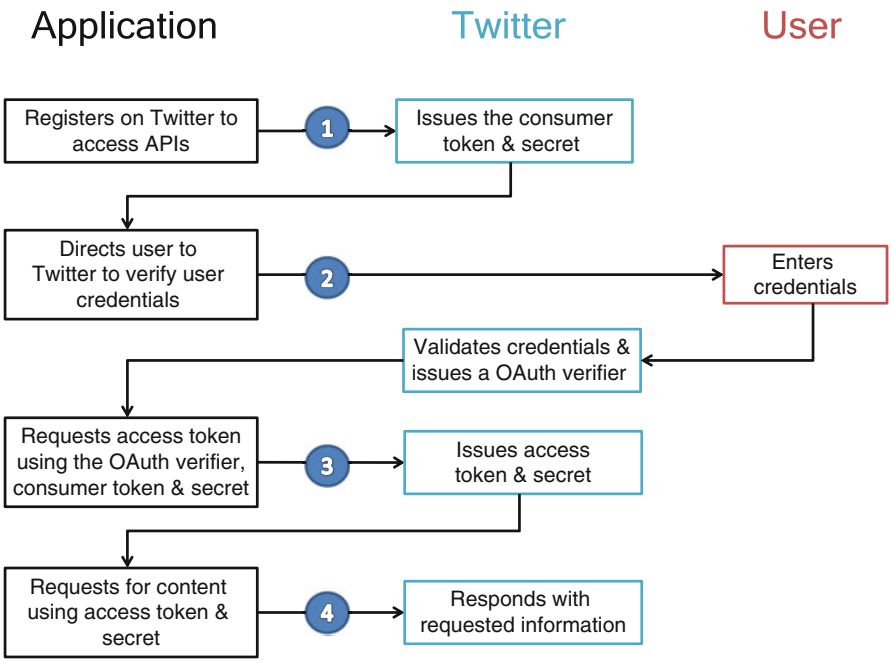

Fig. 2.1 OAuth workflow

3. The user provides this PIN to the application. The application uses the PIN to request an "Access Token" and "Access Secret" unique to the user.
4. Using the "Access Token" and "Access Secret", the application authenticates the user on Twitter and issues API calls on behalf of the user.

The "Access Token" and "Access Secret" for a user do not change and can be cached by the application for future requests. Thus, this process only needs to be performed once, and it can be easily accomplished using the method *GetUserAccessKeySecret* in Listing 2.1.

2.2 Collecting a User's Information

On Twitter, users create profiles to describe themselves to other users on Twitter. A user's profile is a rich source of information about him. An example of a Twitter user's profile is presented in Fig. 2.2. Following distinct pieces of information regarding a user's Twitter profile can be observed in the figure:

Fig. 2.2 An example of a Twitter profile

Listing 2.1 Generating OAuth token for a user

```
public OAuthTokenSecret GetUserAccessKeySecret() {
        . . .
        //Step 1 is performed directly on twitter.com after
            registration.
        //Step 2 User authenticates on twitter.com and generates
             a PIN
        OAuthConsumer consumer = new CommonsHttpOAuthConsumer(
            OAuthUtils.CONSUMER_KEY, OAuthUtils.
            CONSUMER_SECRET);
        OAuthProvider provider = new DefaultOAuthProvider(
            OAuthUtils.REQUEST_TOKEN_URL, OAuthUtils.
            ACCESS_TOKEN_URL, OAuthUtils.AUTHORIZE_URL);
        String authUrl = provider.retrieveRequestToken(consumer,
             OAuth.OUT_OF_BAND);
        //Visit authUrl and enter the PIN in the application
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
        String pin = br.readLine();
        //Step 3 Twitter generates the token and secret using
            the provided PIN
        provider.retrieveAccessToken(consumer,pin);
        String accesstoken = consumer.getToken();
        String accesssecret  = consumer.getTokenSecret();
        OAuthTokenSecret tokensecret = new OAuthTokenSecret(
            accesstoken,accesssecret);
        return tokensecret;
        . . .
}
```

Source: Chapter2/openauthentication/OAuthExample.java

- User's real name (Data Analytics)
- User's Twitter handle(@twtanalyticsbk)
- User's location (Tempe, AZ)
- URL, which typically points to a more detailed profile of the user on an external website (tweettracker.fulton.asu.edu/tda)
- Textual description of the user and his interests (Twitter Data Analytics is a book for...)
- User's network activity information on Twitter (1 follower and following 6 friends)
- Number of Tweets published by the user (1 Tweet)
- Verified mark if the identity of the user has been externally verified by Twitter
- Profile creation date

Listing 2.2 Using Twitter API to fetch a user's profile

```
public JSONObject GetProfile(String username) {
        . . .
        // Step 1: Create the API request using the supplied
            username
        URL url = new URL("https://api.twitter.com/1.1/users/
            show.json?screen_name="+username);
        HttpURLConnection huc = (HttpURLConnection) url.
            openConnection();
        huc.setReadTimeout(5000);
        // Step 2: Sign the request using the OAuth Secret
        consumer.sign(huc);
        huc.connect();
        . . .
        /** Step 3: If the requests have been exhausted,
        * then wait until the quota is renewed
        */
        if(huc.getResponseCode()==429) {
                try {
                        huc.disconnect();
                        Thread.sleep(this.GetWaitTime("/users/
                            show/:id"));
                        flag = false;
        . . .
        // Step 4: Retrieve the user's profile from Twitter
        bRead = new BufferedReader(new InputStreamReader((
            InputStream) huc.getContent()));
        . . .
        profile = new JSONObject(content.toString());
        . . .
        return userobj;
}
```

Source: Chapter2/restapi/RESTApiExample.java

Listing 2.3 A sample Twitter user object

```
{
    "location": "Tempe,AZ",
    "default_profile": true,
    "statuses_count": 1,
    "description": "Twitter Data Analytics is a book for
        practitioners and researchers interested in
        investigating Twitter data.",
    "verified": false,
    "name": "DataAnalytics",
    "created_at": "Tue Mar 12 18:43:47 +0000 2013",
    "followers_count": 1,
    "geo_enabled": false,
    "url": "http://t.co/HnlG9amZzj",
    "time_zone": "Arizona",
    "friends_count": 6,
    "screen_name": "twtanalyticsbk",
   //Other user fields
    . . .
}
```

Using the API *users/show*,[5] a user's profile information can be retrieved using the method *GetProfile*. The method is presented in Listing 2.2. It accepts a valid `username` as a parameter and fetches the user's Twitter profile.

Key Parameters: Each user on Twitter is associated with a unique id and a unique Twitter handle which can be used to retrieve his profile. A user's Twitter handle, also called their screen name (`screen_name`), or the Twitter ID of the user (`user_id`), is mandatory. A typical user object is formatted as in Listing 2.3.

Rate Limit: A maximum of 180 API calls per single user and 180 API calls from a single application are accepted within a single rate limit window.

Note: User information is generally included when Tweets are fetched from Twitter. Although the Streaming API does not have a specific endpoint to retrieve user profile information, it can be obtained from the Tweets fetched using the API.

2.3 Collecting a User's Network

A user's network consists of his connections on Twitter. Twitter is a directed network and there are two types of connections between users. In Fig. 2.3, we can observe an example of the nature of these edges. John follows Alice, therefore John is Alice's follower. Alice follows Peter, hence Peter is a friend of Alice.

[5] https://dev.twitter.com/docs/api/1.1/get/users/show

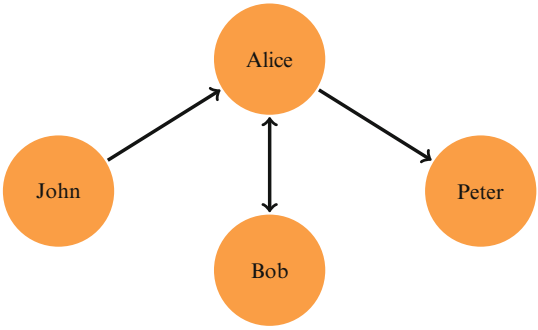

Fig. 2.3 An example of a Twitter network with different types of edges

Listing 2.4 Using the Twitter API to fetch the followers of a user

```
public JSONArray GetFollowers(String username) {
        . . .
        // Step 1: Create the API request using the supplied
            username
        URL url = new URL("https://api.twitter.com/1.1/followers
            /list.json?screen_name="+username+"&cursor="
            + cursor);
        HttpURLConnection huc = (HttpURLConnection) url.
            openConnection();
        huc.setReadTimeout(5000);
        // Step 2: Sign the request using the OAuth Secret
        Consumer.sign(huc);
        huc.connect();
        . . .
        /** Step 3: If the requests have been exhausted,
          * then wait until the quota is renewed
          */
        if(huc.getResponseCode()==429) {
                try {
                        Thread.sleep(this.GetWaitTime("/
                            followers/list"));
                } catch (InterruptedException ex) {
                        Logger.getLogger(RESTApiExample.class.
                            getName()).log(Level.SEVERE, null,
                            ex);
                }
        }
        // Step 4: Retrieve the followers list from Twitter
        bRead = new BufferedReader(new InputStreamReader((
            InputStream) huc.getContent()));
        StringBuilder content = new StringBuilder();
        String temp = "";
        while((temp = bRead.readLine())!=null) {
                content.append(temp);
        }
        try {
```

```
                JSONObject jobj = new JSONObject(content.
                    toString());
                // Step 5: Retrieve the token for the next
                    request
                cursor = jobj.getLong("next_cursor");
                JSONArray idlist = jobj.getJSONArray("users");
                for(int i=0;i<idlist.length();i++) {
                        followers.put(idlist.getJSONObject(i));
                }
        . . .
        return followers;
}
```

Source: Chapter2/restapi/RESTApiExample.java

2.3.1 Collecting the Followers of a User

The followers of a user can be crawled from Twitter using the endpoint *followers/list*,[6] by employing the method *GetFollowers* summarized in Listing 2.4. The response from Twitter consists of an array of user profile objects such as the one described in Listing 2.3

Key Parameters: `screen_name` or `user_id` is mandatory to access the API. Each request returns a maximum of 15 followers of the specified user in the form of a Twitter User object. The parameter "cursor" can be used to paginate through the results. Each request returns the cursor for use in the request for the next page.

Rate Limit: A maximum of 15 API calls from a user and 30 API calls from an application are allowed within a rate limit window.

2.3.2 Collecting the Friends of a User

The friends of a user can be crawled using the Twitter API *friends/list*[7] by employing the method *GetFriends*, which is summarized in Listing 2.5. The method constructs a call to the API and takes a valid Twitter `username` as the parameter. It uses the cursor to retrieve all the friends of a user and if the API limit is reached, it will wait until the quota has been renewed.

Key Parameters: As with the followers API, a valid `screen_name` or `user_id` is mandatory. Each request returns a list of 20 friends of a user as Twitter User objects. The parameter "cursor" can be used to paginate through the results. Each request returns the cursor to be used in the request for the next page.

[6] https://dev.twitter.com/docs/api/1.1/get/followers/list

[7] https://dev.twitter.com/docs/api/1.1/get/friends/list

Listing 2.5 Using the Twitter API to fetch the friends of a user

```
public JSONArray GetFriends(String username) {
        . . .
        JSONArray friends = new JSONArray();
        // Step 1: Create the API request using the supplied
            username
        URL url = new URL("https://api.twitter.com/1.1/friends/
            list.json?screen_name="+username+"&cursor="+cursor);
        HttpURLConnection huc = (HttpURLConnection) url.
            openConnection();
        huc.setReadTimeout(5000);
        // Step 2: Sign the request using the OAuth Secret
        Consumer.sign(huc);
        huc.connect();
        . . .
        /** Step 3: If the requests have been exhausted,
          * then wait until the quota is renewed
          */
        if(huc.getResponseCode()==429) {
                try {
                        Thread.sleep(this.GetWaitTime("/friends/
                            list"));
                } catch (InterruptedException ex) {
                        Logger.getLogger(RESTApiExample.class.
                            getName()).log(Level.SEVERE, null,
                            ex);
                }
        }
        // Step 4: Retrieve the friends list from Twitter
        bRead = new BufferedReader(new InputStreamReader((
            InputStream) huc.getContent()));
        . . .
        JSONObject jobj = new JSONObject(content.toString());
        // Step 5: Retrieve the token for the next request
        cursor = jobj.getLong("next_cursor");
        JSONArray userlist = jobj.getJSONArray("users");
        for(int i=0;i<userlist.length();i++) {
                friends.put(userlist.get(i));
        }
        . . .
        return friends;
}
```

Source: Chapter2/restapi/RESTApiExample.java

Rate Limit: A maximum of 15 calls from a user and 30 API calls from an application are allowed within a rate limit window.

2.4 Collecting a User's Tweets

A Twitter user's Tweets are also known as status messages. A Tweet can be at most 140 characters in length. Tweets can be published using a wide range of mobile and desktop clients and through the use of Twitter API. A special kind of Tweet is the retweet, which is created when one user reposts the Tweet of another user. We will discuss the utility of retweets in greater detail in Chaps. 4 and 5.

A user's Tweets can be retrieved using both the REST and the Streaming API.

2.4.1 REST API

We can access a user's Tweets by using *statuses/user_timeline*[8] from the REST APIs. Using this API, one can retrieve 3,200 of the most recent Tweets published by a user including retweets. The API returns Twitter "Tweet" objects shown in Listing 2.6.

An example describing the process to access this API can be found in the *GetStatuses* method summarized in Listing 2.7.

Key Parameters: We can retrieve 200 Tweets on each page we collect. The parameter `max_id` is used to paginate through the Tweets of a user. To retrieve the next page we use the ID of the oldest Tweet in the list as the value of this parameter in the subsequent request. Then, the API will retrieve only those Tweets whose IDs are below the supplied value.

Rate Limit: An application is allowed 300 requests within a rate limit window and up to 180 requests can be made using the credentials of a user.

Listing 2.6 An example of Twitter Tweet object

```
{
    "text": "This is the first tweet.",
    "lang": "en",
    "id": 352914247774248960,
    "source": "web",
    "retweet_count": 0,
    "created_at": "Thu Jul 04 22:18:08 +0000 2013",
    //Other Tweet fields
    . . .
    "place": {
        "place_type": "city",
        "name": "Tempe",
        "country_code": "US",
        "url": "https://api.twitter.com/1.1/geo/id/7
            cb7440bcf83d464.json",
        "country": "United States",
```

[8] https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

```
        "full_name": "Tempe, AZ",
        //Other place fields
        . . .
    },
    "user": {
        //User Information in the form of Twitter user object
        . . .
    }
}
```

Listing 2.7 Using the Twitter API to fetch the Tweets of a user

```
public JSONArray GetStatuses(String username) {
        . . .
        // Step 1: Create the API request using the supplied
            username
        // Use (max_id-1) to avoid getting redundant Tweets.
        url = new URL("https://api.twitter.com/1.1/statuses/
            user_timeline.json?screen_name=" + username+"&
            include_rts="+include_rts+"&count="+tweetcount+"&
            max_id="+(maxid-1));
        HttpURLConnection huc = (HttpURLConnection) url.
            openConnection();
        huc.setReadTimeout(5000);
        // Step 2: Sign the request using the OAuth Secret
        Consumer.sign(huc);
        /** Step 3: If the requests have been exhausted,
           * then wait until the quota is renewed  */
        . . .
        //Step 4: Retrieve the Tweets from Twitter
        bRead = new BufferedReader(new InputStreamReader((
            InputStream) huc.getInputStream()));
        . . .
        for(int i=0;i<statusarr.length();i++) {
                JSONObject jobj = statusarr.getJSONObject(i);
                statuses.put(jobj);
                // Step 5: Get the id of the oldest Tweet ID as
                    max_id to retrieve the next batch of Tweets
                if(!jobj.isNull("id")) {
                        maxid = jobj.getLong("id");
        . . .
    return statuses;
}
```

Source: Chapter2/restapi/RESTApiExample.java

2.4.2 Streaming API

Specifically, the *statuses/filter*[9] API provides a constant stream of public Tweets published by a user. Using the method *CreateStreamingConnection* summarized in Listing 2.8, we can create a POST request to the API and fetch the search results as a stream. The parameters are added to the request by reading through a list of userids using the method *CreateRequestBody*, which is summarized in Listing 2.9.

Listing 2.8 Using the Streaming API to fetch Tweets

```
public void CreateStreamingConnection(String baseUrl, String
    outFilePath) {
        HttpClient httpClient = new DefaultHttpClient();
        httpClient.getParams().setParameter(CoreConnectionPNames
            .CONNECTION_TIMEOUT, new Integer(90000));
        //Step 1: Initialize OAuth Consumer
        OAuthConsumer consumer = new CommonsHttpOAuthConsumer(
            OAuthUtils.CONSUMER_KEY,OAuthUtils.CONSUMER_SECRET);
        consumer.setTokenWithSecret(OAuthToken.getAccessToken(),
            OAuthToken.getAccessSecret());
        //Step 2: Create a new HTTP POST request and set
            parameters
        HttpPost httppost = new HttpPost(baseUrl);
        try {
             httppost.setEntity(new UrlEncodedFormEntity(
                 CreateRequestBody(), "UTF-8"));
            . . .
             //Step 3: Sign the request
             consumer.sign(httppost);
             . . .
             HttpResponse response;
             InputStream is = null;
             try {
                  //Step 4: Connect to the API
                  response = httpClient.execute(httppost);
                  . . .
                  HttpEntity entity = response.getEntity();
                  try {
               is = entity.getContent();
                        . . .
                        //Step 5: Process the incoming Tweet
                            Stream
                        this.ProcessTwitterStream(is, outFilePath
                            );
                        . . .
}
```

Source: Chapter2/streamingapi/StreamingApiExample.java

[9] https://dev.twitter.com/docs/api/1.1/post/statuses/filter

Listing 2.9 Adding parameters to the Streaming API

```
private List<NameValuePair> CreateRequestBody() {
        List<NameValuePair> params = new ArrayList<NameValuePair
            >();
        if(Userids != null&&Userids.size()>0) {
                //Add userids
                params.add(CreateNameValuePair("follow",
                   Userids));
        }
        if (Geoboxes != null&&Geoboxes.size()>0) {
                //Add geographic bounding boxes
                params.add(CreateNameValuePair("locations",
                    Geoboxes));
        }
        if (Keywords != null&&Keywords.size()>0) {
                //Add keywords/hashtags/phrases
                params.add(CreateNameValuePair("track",
                     Keywords));
        }
        return params;
}
```

Source: Chapter2/streamingapi/StreamingApiExample.java

Key Parameters: The `follow`[10] parameter can be used to specify the userids of 5,000 users as a comma separated list.

Rate Limit: Rate limiting works differently in the Streaming API. In each connection an application is allowed to submit up to 5,000 Twitter userids. Only public Tweets published by the user can be captured using this API.

2.5 Collecting Search Results

Search on Twitter is facilitated through the use of parameters. Acceptable parameter values for search include keywords, hashtags, phrases, geographic regions, and usernames or userids. Twitter search is quite powerful and is accessible by both the REST and the Streaming APIs. There are certain subtle differences when using each API to retrieve search results.

2.5.1 REST API

Twitter provides the *search/tweets* API to facilitate searching the Tweets. The search API takes words as queries and multiple queries can be combined as a comma separated list. Tweets from the previous 10 days can be searched using this API.

[10] https://dev.twitter.com/docs/streaming-apis/parameters#follow

Listing 2.10 Searching for Tweets using the REST API

```
public JSONArray GetSearchResults(String query) {
   try {
         // Step 1:
         String URL_PARAM_SEPERATOR = "&";
         StringBuilder url = new StringBuilder();
         url.append("https://api.twitter.com/1.1/search/tweets.
             json?q=");
         //query needs to be encoded
         url.append(URLEncoder.encode(query, "UTF-8"));
         url.append(URL_PARAM_SEPERATOR);
         url.append("count=100");
         URL navurl = new URL(url.toString());
         HttpURLConnection huc = (HttpURLConnection) navurl.
             openConnection();
         huc.setReadTimeout(5000);
         Consumer.sign(huc);
         huc.connect();
         . . .
         // Step 2: Read the retrieved search results
         BufferedReader bRead = new BufferedReader(new
             InputStreamReader((InputStream) huc.getInputStream()
             ));
         String temp;
         StringBuilder page = new StringBuilder();
         while( (temp = bRead.readLine())!=null) {
                page.append(temp);
         }
         JSONTokener jsonTokener = new JSONTokener(page.toString
             ());
         try{
            JSONObject json = new JSONObject(jsonTokener);
            //Step 4: Extract the Tweet objects as an array
            JSONArray results = json.getJSONArray("statuses");
            return results;
            . . .
}
```

Source: Chapter2/restapi/RESTApiExample.java

Requests to the API can be made using the method *GetSearchResults* presented in Listing 2.10. Input to the function is a keyword or a list of keywords in the form of an OR query. The function returns an array of Tweet objects.

Key Parameters: `result_type` parameter can be used to select between the top ranked Tweets, the latest Tweets, or a combination of the two types of search results matching the query. The parameters `max_id` and `since_id` can be used to paginate through the results, as in the previous API discussions.

Rate Limit: An application can make a total of 450 requests and up to 180 requests from a single authenticated user within a rate limit window.

2.5.2 *Streaming API*

Using the Streaming API, we can search for keywords, hashtags, userids, and geographic bounding boxes simultaneously. The *filter* API facilitates this search and provides a continuous stream of Tweets matching the search criteria. POST method is preferred while creating this request because when using the GET method to retrieve the results, long URLs might be truncated. Listings 2.8 and 2.9 describe how to connect to the Streaming API with the supplied parameters.

Listing 2.11 Processing the streaming search results

```
public void ProcessTwitterStream(InputStream is, String
    outFilePath) {
      BufferedWriter bwrite = null;
      try {
        /** A connection to the streaming API is already
        *  created and the response is  contained in
        *  the InpuStream
        */
           JSONTokener jsonTokener = new JSONTokener(new
               InputStreamReader(is, ``UTF-8''));
           ArrayList<JSONObject> rawtweets = new ArrayList<
               JSONObject>();
           int nooftweetsuploaded = 0;
        //Step 1: Read until the stream is exhausted
           while(true) {
           try {
                JSONObject temp = new JSONObject(jsonTokener);
                rawtweets.add(temp);
                if (rawtweets.size() >= RECORDS_TO_PROCESS){
                Calendar cal = Calendar.getInstance();
                String filename = outFilePath + ``tweets_'' +
                    cal.getTimeInMillis() + ``.json'';
                     //Step 2: Periodically write the
                         processed Tweets to a file
                     bwrite = new BufferedWriter(new
                         OutputStreamWriter(new
                         FileOutputStream(filename),
                         ``UTF-8''));
                     nooftweetsuploaded+=RECORDS_TO_PROCESS;
                     for (JSONObject jobj : rawtweets) {
                  bwrite.write(jobj.toString());
                     bwrite.newLine();
                     }
                     bwrite.close();
                     rawtweets.clear();
                . . .
}
```

Source: Chapter2/streamingapi/StreamingApiExample.java

In method *ProcessTwitterStream*, as in Listing 2.11, we show how the incoming stream is processed. The input is read in the form of a continuous stream and

each Tweet is written to a file periodically. This behavior can be modified as per the requirement of the application, such as storing and indexing the Tweets in a database. More discussion on the storage and indexing of Tweets will follow in Chap. 3.

Key Parameters: There are three key parameters:

- `follow`: a comma-separated list of userids to follow. Twitter returns all of their public Tweets in the stream.
- `track`: a comma-separated list of keywords to track. Multiple keywords are provided as a comma separated list.
- `locations`: a comma-separated list of geographic bounding box containing the coordinates of the southwest point and the northeast point as (longitude, latitude) pairs.

Rate Limit: Streaming APIs limit the number of parameters which can be supplied in one request. Up to 400 keywords, 25 geographic bounding boxes and 5,000 userids can be provided in one request. In addition, the API returns all matching documents up to a volume equal to the streaming cap. This cap is currently set to 1% of the total current volume of Tweets published on Twitter.

2.6 Strategies to Identify the Location of a Tweet

Location information on Twitter is available from two different sources:

- Geotagging information: Users can optionally choose to provide location information for the Tweets they publish. This information can be highly accurate if the Tweet was published using a smartphone with GPS capabilities.
- Profile of the user: User location can be extracted from the location field in the user's profile. The information in the location field itself can be extracted using the APIs discussed above.

Approximately 1% of all Tweets published on Twitter are geolocated. This is a very small portion of the Tweets, and it is often necessary to use the profile information to determine the Tweet's location. This information can be used in different visualizations as you will see in Chap. 5. The location string obtained from the user's profile must first be translated into geographic coordinates. Typically, a gazetteer is used to perform this task. A gazetteer takes a location string as input, and returns the coordinates of the location that best correspond to the string. The granularity of the location is generally coarse. For example, in the case of large regions, such as cities, this is usually the center of the city. There are several online gazetteers which provide this service, including Bing™, Google™, and MapQuest™. In our example, we will use the Nominatim service from MapQuest[11]

[11] http://developer.mapquest.com/web/products/open/nominatim

Listing 2.12 Translating location string into coordinates

```
public Location TranslateLoc(String loc) {
     if(loc!=null&&!loc.isEmpty()) {
        String encodedLoc="";
        try {
        // Step 1: Encode the location name
                encodedLoc = URLEncoder.encode(loc, "UTF-8");
        . . .
        /** Step 2: Create a get request to MapQuest API with
            the
          * name of the location
          */
        String url= "http://open.mapquestapi.com/nominatim/v1/
            search?q="+encodedLoc+"&format=json";
        String page = ReadHTML(url);
        if(page!=null) {
                try{
                        JSONArray results = new JSONArray(page);
                        if(results.length()>0) {
                                //Step 3: Read and extract the
                                    coordinates of the location
                                    as a JSONObject
                                Location loca = new Location(
                                    results.getJSONObject(0).
                                    getDouble("lat"),results.
                                    getJSONObject(0).getDouble("
                                    lon"));
                                return loca;
        . . .
}
```

Source: Chapter2/location/LocationTranslationExample.java

to demonstrate this process. In Listing 2.12, a summary of the method *TranslateLoc* is provided, which is defined in the class *LocationTranslateExample*. The response is provided in JSON, from which the coordinates can be easily extracted. If the service is unable to find a match, it will return (0,0) as the coordinates.

2.7 Obtaining Data via Resellers

The rate limitations of Twitter APIs can be too restrictive for certain types of applications. To satisfy such requirements, Twitter Firehose provides access to 100% of the public Tweets on Twitter at a price. Firehose data can be purchased through third party resellers of Twitter data. At the time of writing of this book, there are three resellers of data, each of which provide different levels of access. In addition to Twitter data some of them also provide data from other social media platforms, which might be useful while building social media based systems. These include the following:

- DataSift™[12] – provides access to past data as well as streaming data
- GNIP™[13] – provides access to streaming data only
- Topsy™[14] – provides access to past data only

2.8 Further Reading

Full documentation of v1.1 of the Twitter API can be found at [1]. It also contains the most up-to-date and detailed information on the rate limits applicable to individual APIs. Twitter HTTP Error Codes & Responses [2] contains a list of HTTP error codes returned by the Twitter APIs. It is a useful resource while debugging applications. The REST API for search accepts several different parameters to facilitate the construction of complex queries. A full list of these along with examples can be found in [4]. The article further clarifies on what is possible using the Search API and explains the best practices for accessing the API. Various libraries exist in most popular programming languages, which encapsulate the complexity of accessing the Twitter API by providing convenient methods. A full list of all available libraries can be found in [3]. Twitter has also released an open source library of their own called the Hosebird, which has been tested to handle firehose streams.

References

1. Twitter. Twitter API v1.1 Documentation. https://dev.twitter.com/docs/api/1.1, 2013. [Online; accessed 19-March-2013].
2. Twitter. Twitter HTTP Error Codes & Responses. https://dev.twitter.com/docs/error-codes-responses, 2013. [Online; accessed 19-March-2013].
3. Twitter. Twitter Libraries. https://dev.twitter.com/docs/twitter-libraries, 2013. [Online; accessed 9-July-2013].
4. Twitter. Using the Twitter Search API. https://dev.twitter.com/docs/using-search, 2013. [Online; accessed 9-July-2013].

[12] http://datasift.com

[13] http://gnip.com

[14] http://topsy.com