Cover your bases with TweetMe4i and JSON

Search for and log twitter messages using open source Java and RPG

Aaron Bartell February 21, 2012

The world of audits is a reality that more financial companies are learning to live with. One of the things being audited are the texts sent to social media sites like Twitter, LinkedIn and Facebook. This tutorial will guide you through how to do Twitter searches from RPG on IBM i and record the results into a DB2 table.

Where it makes sense

I am a big fan of keeping everything in RPG when dealing with programming on IBM i. I figure the less languages and runtimes I have to deal with the more I can focus on meeting business needs vs. focusing on fixing technology. Don't get me wrong, I LOVE learning about new things outside of the RPG landscape - been diving into PhoneGap and Apache Callback lately - but I've learned that the adoption of technology should be done with a full understanding of the longterm ramifications (i.e. it doubles the half-life of information an IT shop needs to keep track of). I digress into this preamble because in this article I am going to show how to query Twitter from RPG with Java in the middle. The reason I am going this route (vs. RPG direct to Twitter) is because there are a handful of things missing from RPG that would make this much more doable - specifically an RPG implementation of the OAuth mechanism of authenticating to Twitter. OAuth is a new(er) mechanism to do authentication from one site or service to another. Learn more about the OAuth authentication protocol from Twitter's perspective.

Twitter4j makes it happen

One of the reasons Java usage is so attractive in today's software eco-system is that it is so prevalent - there are MANY open source Java projects out there that are quite solid and free for you to use. For the purposes of this article, the free and open source Twitter4j project is being used. It should be noted that there are other Java libraries for Twitter. I chose Twitter4j because it seems to be the most promising library (seemed to be active, had a wide breadth of API's, looked to be well thought out and included things like debugging needs, and had a license that works well in commercial situations - Apache License 2.0). If you have no idea why you should be concerned about which license is used on an open source project then you should take a few minutes to read up on licensing. I have learned that not all open source projects are created equal and there is

definitely an art to picking "safe" and good projects. It is something most programmers will need to be knowledgeable on as litigation increases concerning some licenses (i.e. GPLv3).

Twitter4j has many features that allow you to connect with Twitter on a number of fronts, but the purpose of this article is to focus solely on doing Twitter searches. Specifically, we are going to be diving into the Advanced Search capabilities found on this Twitter page so we can learn what profile "aaronbartell" is tweeting within a certain timeframe.

At this point, it would be good to rewind to a few months ago when I released a new open source project named TweetMe4i. The initial code base for TweetMe4i focused on sending status updates to a Twitter account. On the TweetMe4i webpage you can find documentation on how to register for the various aspects of Twitter and also how to configure your IBM i. This article adds to that initial code base, TweetMe4i.java specifically, by providing search functionality as shown in Listing 1 with the `search()` Java method. You can find the full source for TweetMe4i.java in the file provided with this article.

Listing 1. Search Java Code

```
public static String search(String searchStr) {
  String result = "";
  StringBuilder jsonTweet = new StringBuilder();
  try {
    System.setOut(new PrintStream(new FileOutputStream("TweetMe4iLog.txt")));

    Twitter twitter = new TwitterFactory().getInstance();
    QueryResult qryResult = twitter.search(new Query(searchStr));

    boolean firstTime = true;
    jsonTweet.append("{ \"tweets\": [");
    for (Tweet tweet : qryResult.getTweets()) {
      if(!firstTime)
        jsonTweet.append(",");
      else;
        firstTime = false;
    jsonTweet.append(DataObjectFactory.getRawJSON(tweet));
    }
    jsonTweet.append("] }");
    result = resultDS("SUCCESS", jsonTweet.toString());
  } catch (Exception te) {
    result = resultDS("FAIL", "Java Exception:" + stackTraceToString(te));
  }
  return result;
}
```

As you can see, the `search()` Java method receives and returns a String object. An example of a search string is as follows:

```
"from:aaronbartell since:2011-11-01 until:2001-11-30"
```

The basic concept of the `search()` Java method is to call Twitter with the search string and then return the result of that search as a JSON string to the calling program. In this case, the calling program is written in RPG, which we will be reviewing in a next section. JSON is utilized as the result set return mechanism because it is less laborious to process on the RPG end of things -

especially when we have access to a free and open source JSON parser and serializer in RPG - more on that later.

In the `search()` method, you'll notice the code making a call to `System.setOut()`. This call is done so that the program can redirect `System.out` messages to a stream file to make my debugging efforts significantly easier. Please note that the code doesn't specify a fully qualified path with `TweetMe4iLog.txt`. This means that the stream file will be written to the current directory. You can view your current directly by simply entering `WRKLNK` onto the IBM i command line with no other options.

Next in Listing 1, the code obtains a new instance of a `Twitter` object using the `TwitterFactory.getInstance()` call. The code then invokes `twitter.search()` and passes the search string that was provided in the call to this method. Underneath the covers an HTTP request is being made to Twitter.com to submit the search request and retrieve the results back into the `qryResult` object. The program then iterates over the `qryResult` object and starts composing the JSON string that will be passed back to the RPG program. The reason that the composition logic hard codes some of the JSON string is because an enveloping structure is required for the JSON value provided by the call to `DataObjectFactory.getRawJSON(tweet)` which is used to get the JSON representation of the twitter result set. A sample of the resulting JSON output is shown in Listing 2.

To make `DataObjectFactory.getRawJSON(tweet)` work successfully, you need to add an entry to the existing `twitter4j.properties` file, which if you installed `TweetMe4i` from `http://mowyourlawn.com/tweetme4i.html` it will exist at location `/java/tweetme4i/twitter4j.properties`. The addition to the `twitter4j.properties` file is `jsonStoreEnabled=true`.

Listing 2. JSON Result from search () method

```
{
   "tweets":[
      {
         "text":"Opps, wrong url on that last one ( though a great Christmas
            song :-)  Try this one: http://t.co/p1w3x4jt",
         "from_user_id":44956334,
         "id":144505404985057280,
         "from_user":"aaronbartell",
         "created_at":"Wed, 07 Dec 2011 19:55:47 +0000",
         "metadata":{
            "result_type":"recent"
         }
      },
      {
         "text":"Give feedback for next #RPGNextGen 2.0 features
            http://t.co/rwZL1KH2  #opensource #free #RPG #IBMi editor
            #eclipse based",
         "from_user_id":44956334,
         "id":144504285328195584,
         "from_user":"aaronbartell",
         "created_at":"Wed, 07 Dec 2011 19:51:21 +0000",
         "metadata":{
            "result_type":"recent"
         }
      },
      {
         "text":"RT @SystemiNetwork: Maxed Out: New IBM Tools Uses POWER
```

```
            to Crush Commodity Server Architectures: .http://bit.ly/u36umU
           #IBMi",
         "from_user_id":44956334,
         "from_user_name":"Aaron Bartell",
         "id":143789937253294080,
         "from_user_id_str":"44956334",
         "from_user":"aaronbartell",
         "created_at":"Mon, 05 Dec 2011 20:32:47 +0000",
         "metadata":{
            "result_type":"recent"
         }
      }
   ]
}
```

If this is your first exposure to JSON, then you can learn more about it here. JSON stands for `Javascript Object Notation` and was made popular in the web browser programming world as a way to easily serialize and de-serialize Javascript objects so they could be communicated to and from the server. For example, in the past I have used it extensively with OpenRPGUI and ExtJS. Lately, I have been extending JSON beyond Javascript scenarios. For example, JSON has used it to communicate with an Android application in my DynaDroid4i efforts where Javascript wasn't involved at all. However, JSON was still used to send messages to and from the IBM i to the Android mobile device.

RPG side of things

Now it's time to move to the RPG side of the solution to see how to invoke the Java `search()` method. Listing 3 shows the mainline of RPG program TM4ISEARCH. Note the full source for TM4ISEARCH can be found in this article's accompanying zip file.

Listing 3. TM4ISEARCH mainline

```
      monitor;
        cmd = 'ADDENVVAR ENVVAR(CLASSPATH) REPLACE(*YES) VALUE(' +
           qte +
           '/java/tweetme4i' +
           ':/java/tweetme4i/TweetMe4i.jar' +
           ':/java/tweetme4i/twitter4j-core-2.2.5.jar' +
           qte + ')';

        QCMDEXC(%trimr(cmd): %len(%trimr(cmd)) );

        cmd = 'ADDENVVAR ENVVAR(QIBM_RPG_JAVA_PROPERTIES) ' +
          'REPLACE(*YES) VALUE(' + qte + '-Djava.version=1.5;' + qte + ')';

        QCMDEXC(%trimr(cmd): %len(%trimr(cmd)) );
      on-error;
      endmon;

      gSrchStr = 'from:aaronbartell since:2011-10-01';
      jStr = TweetMe4i_search( newStr(gSrchStr) );
      gResult = getBytes(jStr);

      if gResult.code = 'SUCCESS';
        jsonToDB(gResult.text);
      endif;
```

In Listing 3 I have bolded the areas that we will be discussing. First we see the `CLASSPATH` environment variable being added to include both `TweetMe4i.jar` and `twitter4j-core-2.2.5.jar`.

The `TweetMe4i.jar` contains the `search()` function discussed earlier and `twitter4j-core-2.2.5.jar` is the latest version of Twitter4j as of this writing. Next, look at the formulation of the search string being applied to the `gSrchStr` variable which is then used on the call to `TweetMe4i_search()`. The RPG prototype for the `TweetMe4i_search` API can be seen in Listing 4.

Listing 4. TweetMe4i_search prototype

```
D TweetMe4i_search...
D                 pr              o   class(*java: jStrConst) static
D                                     extproc(
D                                     *java:
D                                     'com.mowyourlawn.twitter.TweetMe4i':
D                                     'search')
D  pTxt                           o   class(*java: jStrConst) const
```

After the RPG program receives the JSON response back, the last bit of important code in Listing 3 is the call to `jsonToDB()`. The `jsonToDB()` sub procedure can be seen in Listing 5 and its purpose is to take the inputted JSON string, parse it and write the contents to a DB2 table, `TWTHST` (short for Tweet History).

Listing 5. jsonToDB sub procedure

```
 P jsonToDB        b
 D jsontoDB        pi
 D  pStr                      65535a

 D jRoot           S               *
 D jTweets         S               *
 D jObj            S               *
 D tweetCount      S             10i 0
 D x               S             10i 0
  /free

   jRoot = json_parse(%addr(pStr));

   jTweets = json_getArray(jRoot: 'tweets');
   tweetCount = jsona_size(jTweets);

   for x = 0 to (tweetCount - 1);
     jObj = jsona_getObject(jTweets: x);

     if jObj = *null;
       leave;
     endif;

     clear TWTHSTR;
     ID = json_getLong(jObj: 'id');
     chain ID TWTHST;
     if not %found(TWTHST);
       TXT = %str( json_get(jObj: 'text'));
       FRMUSRID = json_getInt(jObj: 'from_user_id');
       FRMUSR = %str( json_get(jObj: 'from_user'));
       CRTDAT = %str( json_get(jObj: 'created_at'));
       write TWTHSTR;
     endif;
   endfor;

   json_dispose(jRoot);

  /end-free
 P                 e
```

This sub procedure introduces the JSON *SRVPGM object from Mihael Schmidt of RPGNextGen.com. All of the sub procedure calls that start with `json_` or `jsona_` are from Mihael's JSON *SRVPGM and the variables that start with a lower case `j` are meant to represent the JSON *SRVPGM's objects (pointers, actually). You should definitely take a look at RPGNextGen.com if you haven't already done so. Mihael has been very busy and generous towards the RPG community with his many open source tools.

The first step in the `jsonToDB` sub procedure is calling the `json_parse()` routine which will take the JSON string and store it in a specialized pointer array under the covers. This operation allows the RPG program to subsequently access the various JSON parts by name. The code uses `json_getArray()` to obtain the `tweets` portion of the array and store reference to it in the `jTweets` pointer - if necessary, go back to Listing 2 for a review of the structure. After obtaining the total number of `tweets` with `jsona_size(),` the code logic starts iterating through the results starting at an index of zero. The open source contributor, Mihael, must have had Java on the mind the day he created this part of the tool - we all know RPG people like to start indexes at 1! The call to `jsona_getObject()` retrieves a `tweets` iteration and stores it in `jObj`. After determining whether or not the `jObj` is null (i.e. was the retrieval successful and did the entry exist), the code starts obtaining the various portions of the JSON `tweets` result using `json_*` API calls and populate them into the DB2 record. If you dig into the `json_*` prototypes you'll find that there are generic "getters" like `json_get(),` and more specific ones like `json_getString()` or `json_getInt()`. Either approach works fine, though the more generic ones require a type-cast as shown (i.e. `%str()`).

The `TWTHST` table definition is shown in Listing 6. Each of the columns in `TWTHST` are occupied with data from Twitter, including the unique key of the table. The `column label` values correspond to the actual name of the JSON entity that Twitter and Twitter4j provided.

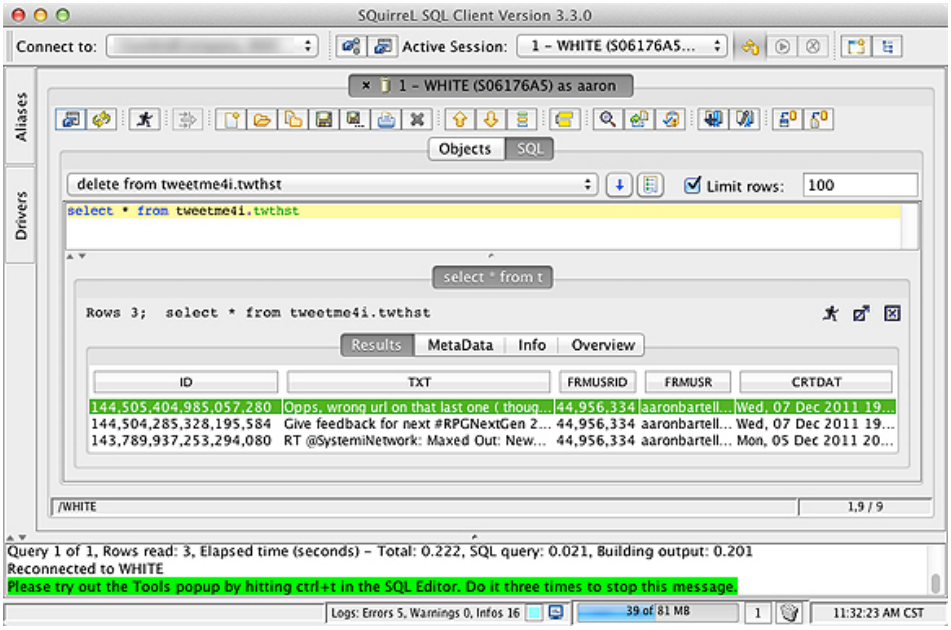
Listing 6. TTHST table definition

```
CREATE TABLE TWEETME4I.TWTHST (
  ID NUMERIC(30,0) NOT NULL DEFAULT 0 ,
  TXT CHAR(140) CCSID 37 NOT NULL DEFAULT '' ,
  FRMUSRID NUMERIC(9, 0) NOT NULL DEFAULT 0 ,
  FRMUSR CHAR(20) CCSID 37 NOT NULL DEFAULT '' ,
  CRTDAT CHAR(31) CCSID 37 NOT NULL DEFAULT '' ,
  PRIMARY KEY( ID ) );

LABEL ON COLUMN TWEET4MEI.TWTHST
(    ID TEXT IS 'ID' ,
 TXT TEXT IS 'TEXT' ,
 FRMUSRID TEXT IS 'FROM_USER_ID' ,
 FRMUSR TEXT IS 'FROM_USER' ,
 CRTDAT TEXT IS 'CREATED_AT' ) ;
```

The last required step in `jsonToDB()` is to clean up all of the allocated memory by calling `json_dispose()`. At this point ,the coding is complete and there are now results in the TWTHST DB2 table as shown in Figure 1 via a screen shot from Squirrel SQL (a great tool to check out if you aren't already using it).

Figure 1. TWTHST Content via Squirrel SQL



Summary

So that's it! You have officially done a Twitter search from your IBM i and stored the results in our beloved DB2. Note that there are also API's for the other social networks like LinkedIn and Facebook, but those offerings also require more permission to see the content that is being sent back and forth (i.e. on Facebook you need to be somebody's friend to see their content). If you would find value in seeing an API for Facebook or LinkedIn, then please email me and I will see what effort it would take to put something together and subsequently write about it.

Resources

- How to use advanced Twitter search
- Twitter
- http://twitter4j.org
- http://json.org
- http://rpgnextgen.com
- http://mowyourlawn.com/tweetme4i.html