Twitter

HISTORY, CONCEPTS, NUTSHELL EXAMPLES WITH (BSF4)OOREXX

Benjamin Berggold
1151964 | 0182 IS PROJECT SEMINAR OF INFORMATION SYSTEMS
SUPERVISOR: AO. UNIV. PROF. DR. RONY G. FLATSCHER
DECEMBER 18, 2015

Declaration

I do solemnly declare that I have written the presented research thesis

Twitter: History, Concepts, Nutshell Examples with (BSF4)OOREXX

by myself without undue help from a second person others and without using such tools other than that specified. Where I have used thoughts from external sources, directly or indirectly, published or unpublished, this is always clearly attributed. Furthermore, I certify that this research thesis or any part of it has not been previously submitted for a degree or any other qualification at the Vienna University of Economics and Business or any other institution in Austria or abroad.

Vienna, the December 7, 2016

_____

Benjamin Berggold

Content

Figures

1. Introduction

This paper first introduces the history of the successful company Twitter, its growth and usage.

Then it dives into the possibilities for developers to communicate with the Twitter Client using the Twitter API. The API itself is explained and also the security mechanism (OAuth) and the limitations are part of the paper.

The programming language used is Rexx, furthermore ooRexx with the extension BSF4ooRexx. This makes it possible to create Java objects from Rexx which get camouflaged as ooRexx objects by BSF4ooRexx.

The last chapter demonstrates the concepts with ten nutshell examples. The code of every example is explained in detail and the results are displayed on screenshots.

2. History

The seminar paper starts with a brief introduction to Twitter's history, growth and usage since the beginning in 2004.

2.1. Creation

Twitter's creation began with the start of Odeo in 2004, it's parent company. Odeo was a podcast publishing and aggregation website which enabled users to create, record and share podcasts with a simple Adobe Flash-based interface. The company was founded by Noah Glass and Evan Williams, both creators of Twitter but the podcasting service had very little public response.

In February 2006 Jack Dorsey, Noah Glass, Evan Williams and Biz Stone did a daylong brainstorming session and discussed the idea of using text messaging to share statuses. The team decided to work on the project and named it "twttr". The name was inspired by Flickr and the American SMS short codes. Also the domain twitter.com was already in use, but the company bought it six months after the launch and renamed the service to Twitter. On March 21 Jack Dorsey sent the first tweet:

> "just setting up my twttr" – Jack Dorsey, March 21, 2006, 9:50 PM

Twitter was now officially set up but only used internally by Odeo employees. Not before July 15 when Om Malik wrote an article about Twitter on GigaOm, Twitter was introduced publicly as the full version. [1]

2.2. Growth

At first Twitter's popularity was little until the South by Southwest Interactive (SXSWi) conference in 2007 took place. The usage increased from 20,000

tweets per day to 60,000 and the reactions from the participants where highly positive. Moreover, Twitter won the festival's Web Award prize with the following message:

> "we'd like to thank you in 140 characters or less. And we just did!"

Since then the company experienced rapid growth. From 100 million tweets posted per quarter in 2008 to 50 million tweets per day in February 2010. Twitter became the third-highest-ranking social networking website in January 2009, ranked by monthly visits on "compete.com". [1]

2.3. Usage

Twitter's usage is rising especially during prominent events. Examples are the 2010 FIFA World Cup with 2,940 tweets per second on June 14 and the 2010 NBA Finals with 3,085 tweets per second on June 17. During specific happenings at these events, like Japan scoring against Cameroon in the first example and after the Los Angeles Lakers victory in the second, the tweets per second were setting a record.

But not only sport events are making users send a tweet. The current record was set after a television screening of the movie "Castle In The Sky" in Japan on August 3, 2013. Twitter users were sending 143,199 tweets per second. The normal tweets per second were about 5,700 tweets on average, meaning the particular spike was 25 times greater.

As of May 2015, Twitter has more than 500 million users, out of which more than 302 million are active users. [1]

3. Tweet

A tweet is an online posting created by a Twitter user. It is also often referred to a "micro-blog" as the posting is limited to 140 characters or less. Originally the purpose of tweets was to answer the question, "What are you doing?". However, tweets can contain any information the user wants to share. Users may subscribe to other users, known as "following". Subscribers are known as "followers", also often called "tweeps", which is a mix of Twitter and peeps (short for "people"). After posting the message, all followers of the user can view it on their Twitter home page. Correspondingly, the users Twitter home page will display the most recent tweets of the users he or she is following. Because of the 140 characters limit it is possible to show several tweets on one page. [2]

Tweets can be posted through various channels:

- The official Twitter website, twitter.com,
- Compatible external applications, e.g. on smartphones or tablets,
- In certain countries via SMS short code.

By default, tweets are publicly visible but users can restrict the visibility to just their followers. [1]

4. Twitter for Developers

This Chapter will introduce the possibilities Twitter offers to developers.

4.1. Twitter API

An application programming interface (API) consists of routines, protocols and

tools for building software applications for various programs and platforms like twitter. It provides building blocks which makes it easier for the programmer to develop a program based of the original program. Using a given programming language, the API makes it possible to develop applications for the existing system. As an example, an app developer for iOS may use the iOS API to interact with the hardware and software of an iPhone.

APIs often are libraries with specifications for routines, data structures, object classes and variables. But they can also only consist simply of remote calls exposed to the API consumers, like twitter's API which is called a RESTful API. [3]

The Twitter API uses representational state transfer (REST) which gives a coordinated set of constraints for a higher-performing and more maintainable architecture. RESTful systems, like twitter's API, communicate over Hypertext Transfer Protocol (HTTP) with its verbs: GET, POST, PUT, DELETE, etc. [4]

When programming with the Twitter API it is important to keep in mind that there are certain limits which restrict the use of the API. For example, the search is limited to 180 queries per 15 minute window. An up-to-date chart with all the limitations the API developer has to consider can be found at "https://dev.twitter.com/rest/public/rate-limits". [5]

4.2. OAuth

Twitter uses OAuth to provide authorized access to its API. Through OAuth account security is increased as users are not required to share their passwords with 3rd party applications. Additionally, many client libraries and example code are compatible with the OAuth implementation, setting a standard. There are two forms of authentication.

First the "application-user authentication" which is the most common form of resource authentication. The signed request identifies the application's identity and also the identity accompanying granted permission of the end-user the API

is making calls on behalf of.

Second the “application-only authentication” where the application makes API requests without a user context. Therefore, any request to the API that require user context, such as posting tweets, will not work. However, the application will be able to, for example pull user timelines, access friends and followers of any account, access lists resources, search tweets or retrieve any user information. [6]

4.3. Twitter Apps

At “apps.twitter.com” it is possible to register for a twitter developer account and create an application for using the twitter API. After creating an app, it is possible to manage the access level from only “read” to “read and write” to “read, write and direct messages”. For the nutshell examples we need the third option as we are sending direct messages. Also the Oath keys are generated on this site. Consumer key, consumer secret, access token and access token secret are displayed on the “apps.twitter.com” website.

5. Used Software

The following chapter gives on overview of the programming languages and libraries used to create the nutshell examples.

5.1. Java

For programming the nutshell examples it is essential to understand how Java works as Java methods are used to communicate with the Twitter API. Therefore, this chapter gives a brief introduction to the programming

language.

The concept behind Java is "write once, run everywhere" which means that compiled Java code is able to run on any platform that supports Java without the need of recompilation. It is a class-based, object-oriented programming language with few implementation dependencies. As of 2015, it is the most popular programming language in use, with a reported 9 million developers. Its syntax derives from C and C++. [7]

The famous "Hello World!" program code looks like the following:

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Figure 1: Hello World!

5.2. ooRexx

Open Object Rexx is an open source, object oriented version of Rexx (REstructured eXtended eXecutor) which was developed by Mike Cowlishaw at IBM. It has an English-like syntax which makes it easy to use, read and learn.

The "Hello World!" program code looks like the following:

```
SAY "Hello World!"
```

Open Object Rexx is nowadays further developed by the Rexx Language Association (RexxLA) which is an independent, non-profit organisation dedicated to promoting the use and understanding of Rexx. [8]

5.3. BSF4ooRexx

The Bean Scripting Framework (BSF) allows interaction with the Java runtime environment for other programming languages. This enables the use of Java

classes and Java objects by them.

The Bean Scripting Framework for Open Object Rexx (BSF4ooRexx) makes it possible to create Java objects from Rexx which get camouflaged as ooRexx objects by BSF4ooRexx. All the needed data type and case conversions between Java and ooRexx are handled by the BSF4ooRexx infrastructure.

For the nutshell examples it is important to note that the line of code "::requires BSF.CLS" is necessary to enable Java support. "The ooRexx package BSF.CLS defines public routines and the ooRexx class BSF which allows creating Java objects from a Rexx program. The returned ooRexx objects are instances of the class BSF and will forward received messages for execution to Java and return any value the Java method returns. If a Java object gets returned, it will get wrapped up as an instance of the ooRexx class BSF."

The Framework is developed by Mag. Dr. Rony G. Flatscher at the Vienna University of Economics and Business. [9]

5.4. Twitter4j

Twitter4J is an unofficial Java library for the Twitter API. The library consists of many Java classes which communicate with the Twitter API, making programming the API with Java very easy. All the following nutshell examples take advantage of the library and demonstrate how it works. [10]

6. Nutshell Examples

Finally the following chapter describes the creation of the ten nutshell examples and explains them in detail.

6.1. Configuration & Setup

The first part of every ".rxj" file created for the nutshell examples consists of the same lines of code. These lines are shown in Figure 2.

```
cb=.bsf~new("twitter4j.conf.ConfigurationBuilder",)
cb~setOAuthConsumerKey("djhqZM2zOwGtMh91glSGD0Tmt")
cb~setOAuthConsumerSecret("eXMldFpprQJQEZDBauYr1vO1c1PokmJnkqCOj33ceq2bnWny8g")
cb~setOAuthAccessToken("1008446732-FyS6bC53dBBf6L7D9ga8fDEggN7Eg8R19z5Bmhy")
cb~setOAuthAccessTokenSecret("dTRVoqHkyddhzjUVfqEHXHjOU8Idf9JDSHzfVFgj0mgw5")

tw=.bsf~new("twitter4j.TwitterFactory",cb~build)~getInstance
```

Figure 2: ConfigurationBuilder

The first line calls the method "twitter4j.conf.ConfigurationBuilder" from the twitter4j library, discussed at the previous chapter. It is stored in the variable "cb" which can now be used to alter the twitter4j configuration with desired settings. There are many settings which can be customized through the builder, important for the nutshell examples are the following four:

1. setOAuthConsumerKey
2. setOAuthConsumerSecret
3. setOAuthAccessToken
4. setOAuthAccessTokenSecret

The four OAuth keys are created by the Twitter API and can be regenerated at any time through the "apps.twitter.com" homepage. Simple "copy and paste" of the codes enables the program to communicate with the twitter client. As described in the Chapter "OAuth" the keys increase security and enable authentication.

After successful authentication one more step is necessary before it is possible to use twitter4j's library. The library has a built in factory method. A factory is an object for creating other objects. It returns an object with a method call "new" and can be seen as an abstraction of a constructor of a class.

With "cb~build" inside the factory method a usable configuration is created

and with "~getInstance" the factory returns an instance associated with the configuration bound to it. The factory instance is then stored in the "tw" variable at the beginning of the line which can now be used to write programs with the twitter4j library.

The following chapters consist of ten nutshell examples which demonstrate the possibilities of the Twitter API.

6.2. updateStatus

The first example describes the simple task of posting a new status on the user's twitter timeline. The method needed is "updateStatus()" which is part of the twitter4j library. We call the method with the factory instance "tw" followed by a "~" and the method name "updateStatus". Inside the parenthesis, after the method name, the text for the timeline is written. In this case the array "arg(1)" is inside as shown in Figure 3 which means it takes the input written after the execution of the program and puts it into the parenthesis after the method name.

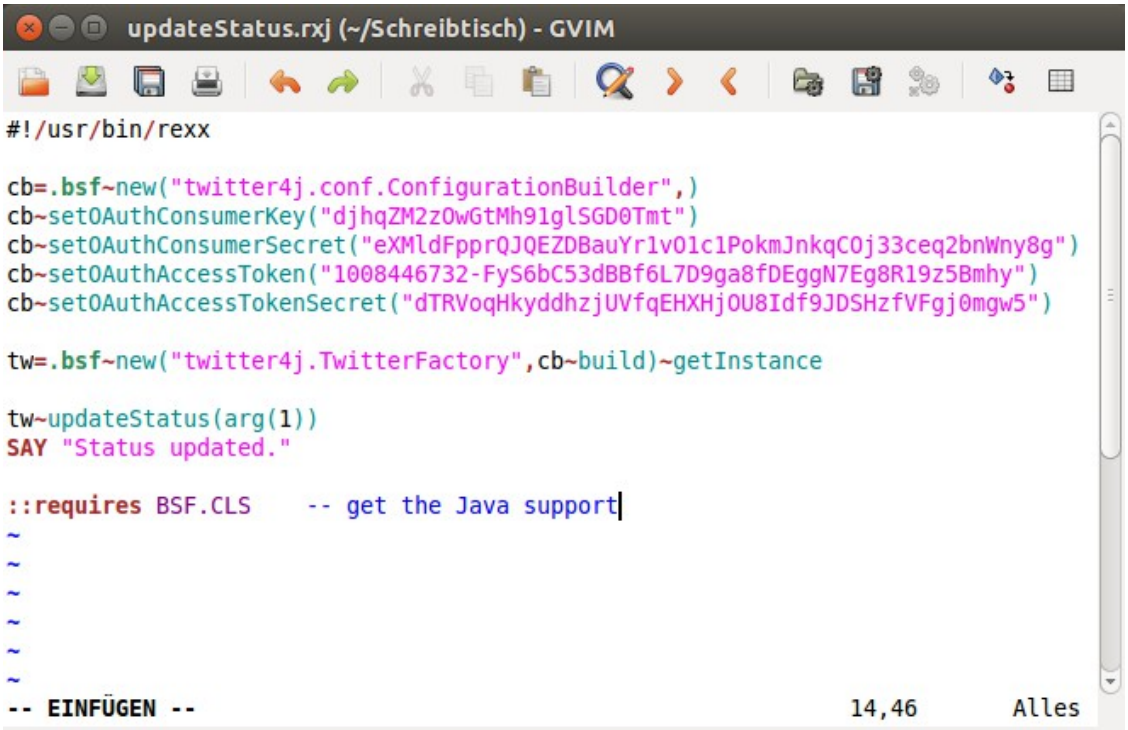

Figure 3: updateStatus

The execution of the program is displayed in Figure 4.

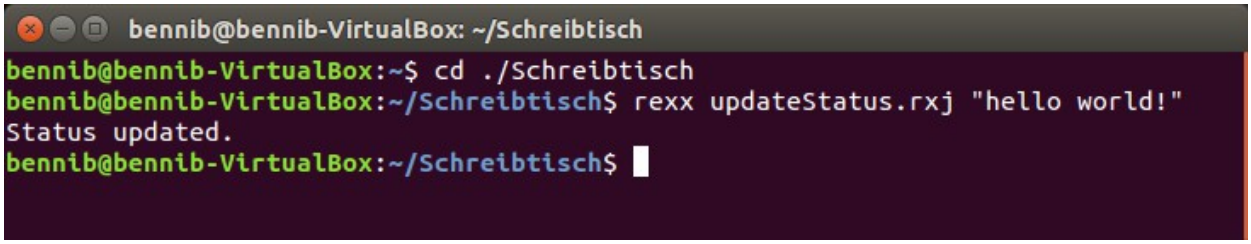

Figure 4: updateStatus Terminal

In this case "hello world!" is stored in the first position of the array "arg" and put into the parenthesis after the method "updateStatus". The entire line of code would look like the following:

tw~updateStatus("hello world!")

The next line is a simple "SAY" command which reports only to the terminal that the status has been updated.

Finally, the result in twitter itself can be seen in Figure 5.

Figure 5: updateStatus Twitter

6.3. Timeline

The second example displays the twitter home timeline in the terminal as a list. The method used is "getHomeTimeLine" from the twitter4j library. Like in the previous example we call the method with the factory instance "tw" followed by a "~" and the method name "getHomeTimeLine". The timeline is then stored into the variable "statuses". We need to put the timeline information into an array to filter the results and create a list. Therefore we store the timeline information into an array with the command "statuses~toArray" stored in the "dir" variable. The next line simply prints the text "Hometimeline:" to the terminal screen. All the code is shown in Figure 6.

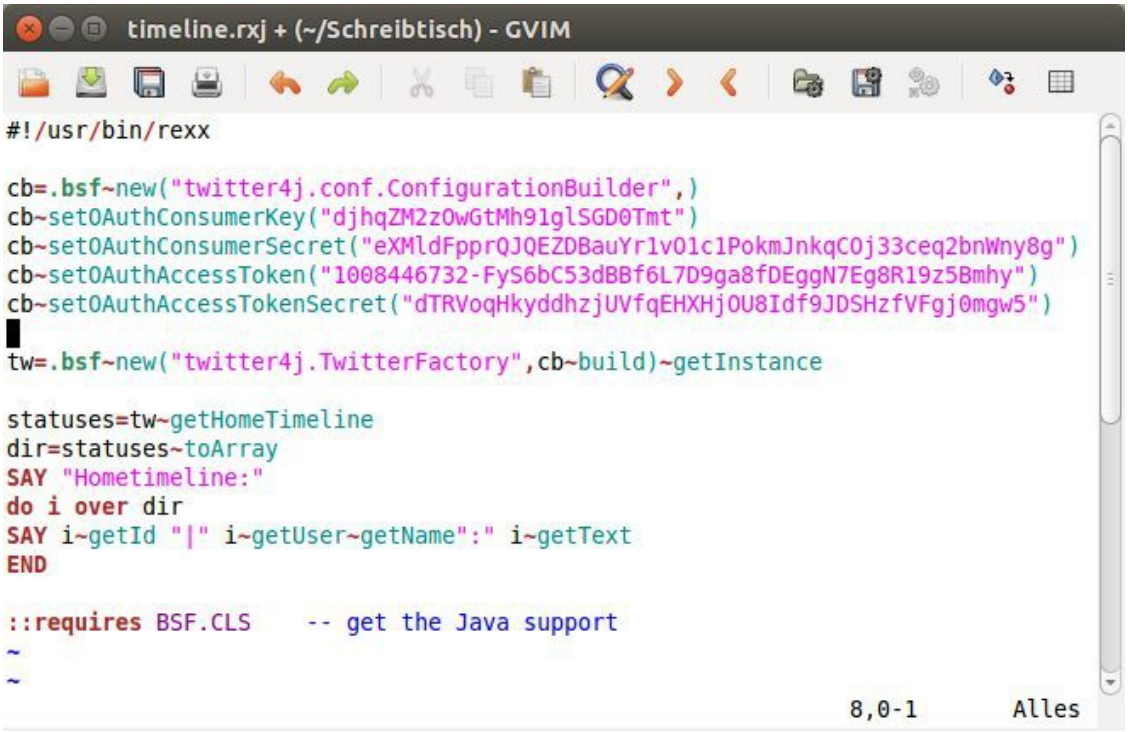

Figure 6: Timeline

Finally, we can loop through the array and display the statuses on the home timeline. This is possible with the "do i over dir" loop which stores every value of the dir array one by one separately into the variable "i" and prints it out in the next line. With the "i~getId" we call the ID of the status, which is a unique number, "i~getUser~getName" returns the username who posted the status and "i~getText" displays the status text. The "END" statement ends the loop and the program is finished.

After executing the program through entering the line “rexx timeline.rxj” into the terminal, the home timeline is displayed in the terminal. The result is shown in Figure 7.

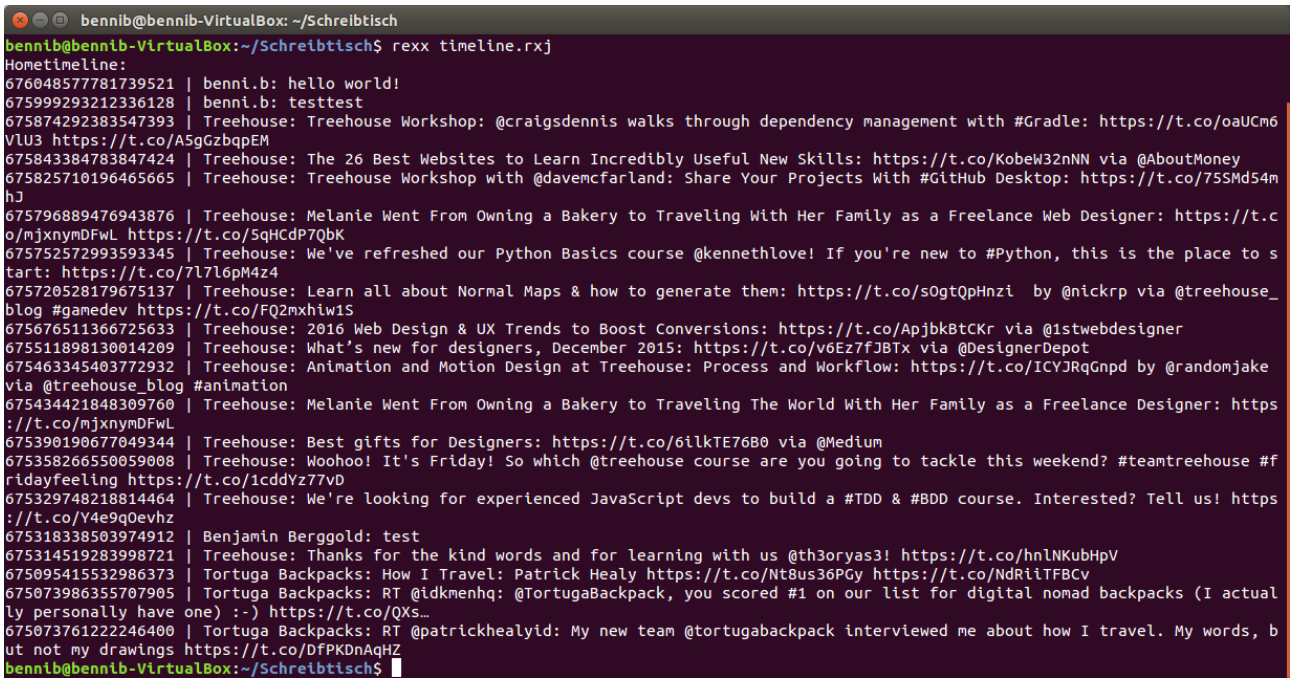

Figure 7: Timeline Terminal

6.4. Favorite

The third example makes it possible to favourite any status. The method needed is “createFavorite” from the twitter4j library. The lines of code are similar to the lines from the “updateStatus” example as we are also using the “arg” array command line input.

As shown in Figure 8, first we are calling the factory instance with “tw” followed by a “~” and the method name “createFavorite()”. After the method name, inside the parenthesis, the ID of the status that gets favorited is written. In the program the ID is written with the execution, after the file name in the terminal, similar to the “updateStatus” example program. The ID is then stored into the first position of the array “arg” and handed inside the parenthesis after the “createFavorite” method.

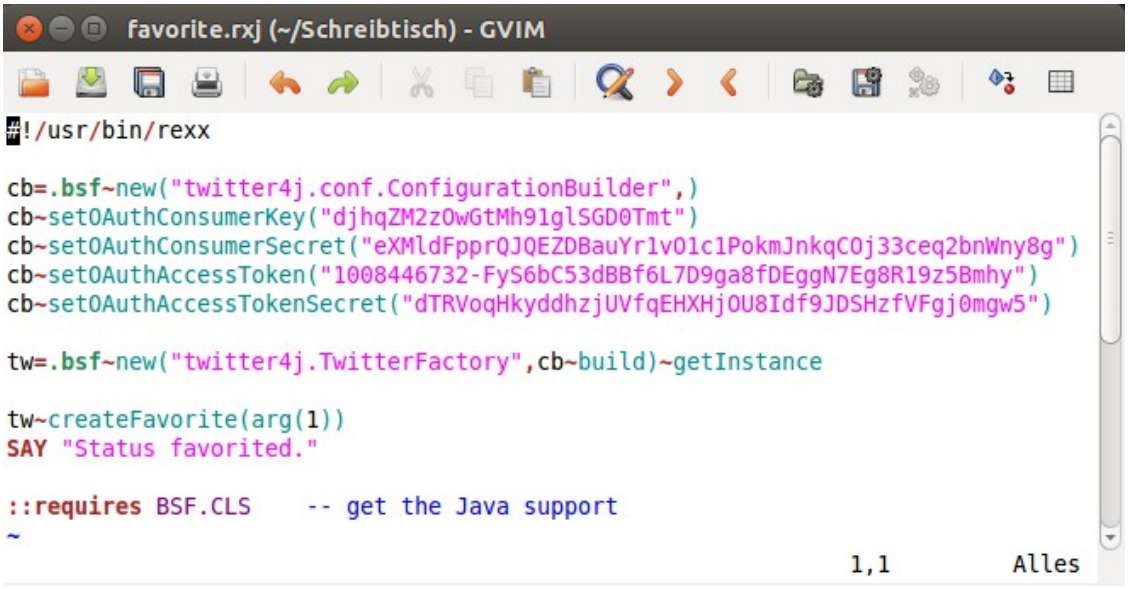

Figure 8: Favorite

The method takes the ID and favourites the twitter status. The entire line of code would look like the following:

tw~createFavorite(676048577781739521)

After that the program runs a simple "SAY" command printing "Status favorited.". The execution in the terminal is displayed in Figure 9.

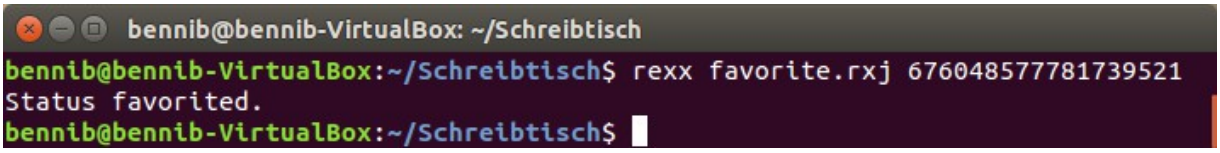

Figure 9: Favorite Terminal

Finally, the result in twitter itself can be seen in Figure 10. The favourited status is marked with a red heart.

Figure 10: Favorite Twitter

The fourth example program lists all the favourited statuses of the user in the terminal. The method needed is "getFavorites" from the twitter4j library. As always we call the method with the factory instance "tw" followed by a "~" and

the method name "getFavorites". The favourites are then stored into the variable "favorites". Then an array is needed to filter the results and create a list. We do this with storing the favourites into an array with the command "favorites~toArray" and save it in the "dir" variable. All lines of code are shown in Figure 11.

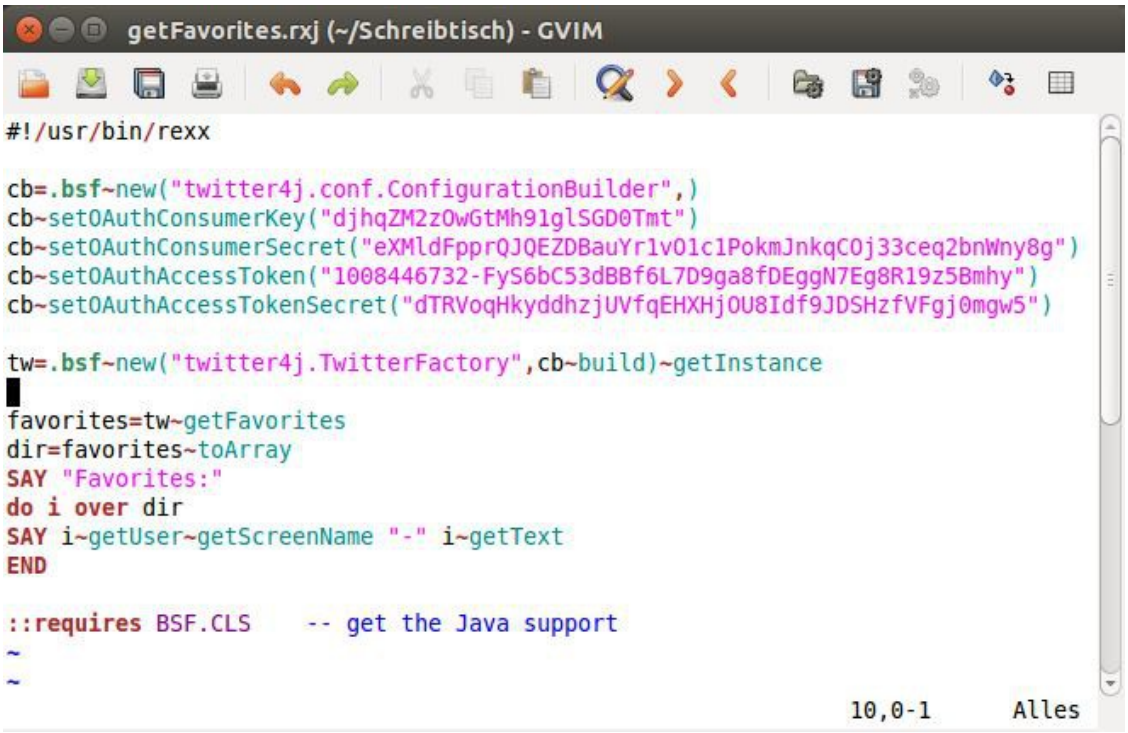

Figure 11: getFavorites

The next line prints a simple text to the terminal screen saying "Favorites:".

In the following three lines of code we create a "do over" loop, similar to the one in the timeline example. Every value of the array "dir" is stored into the variable "i" separately and is printed out on the terminal. "i~getUser~getScreenName" returns the username of the favourited status and "i~getText" displays the status text. The "END" statement ends the loop and the program is finished.

After executing the program with typing the line "rexx getFavorites.rxj" into the terminal, the favourites are displayed as a list in the terminal. The result is shown in Figure 12.

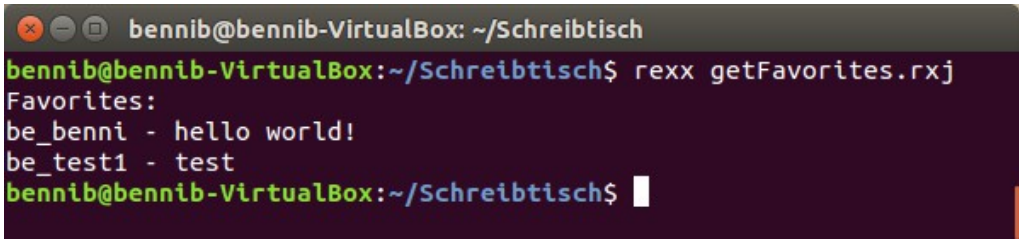

Figure 12: getFavorites Terminal

In twitter itself the favourites can be seen in Figure 13.

Figure 13: getFavorites Twitter

6.6. destroyStatus

In the fifth example the status posted in the first example gets deleted. The method needed is "destroyStatus()" from the twitter4j library. We call the method with the factory instance "tw" followed by a "~" and the method name "destroyStatus". After the method name, inside the parethesis, the ID of the status which should get deleted is entered. Again we do this with the "arg" array and take the input directly from the terminal execution. As shown in Figure 14 we put the variable "arg(1)" inside the parenthesis.

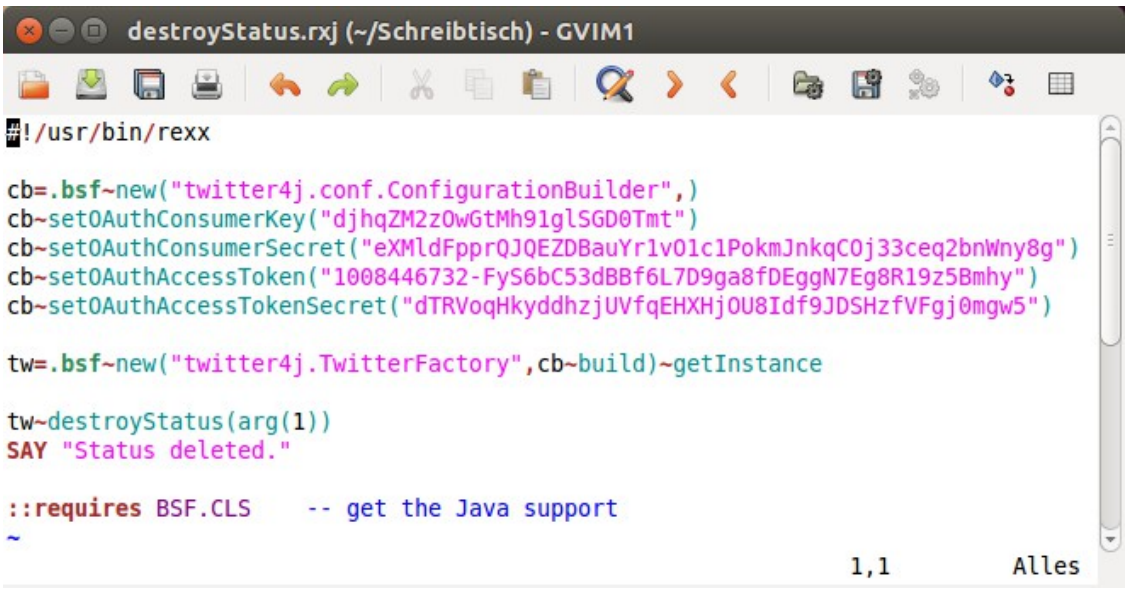

Figure 14: destroyStatus

The execution of the program with the ID of the status at the end of the line is shown in Figure 15. In this case the status ID "675999293212336128" is stored in the first position of the array "arg" and put into the parenthesis after the method name "destroyStatus".

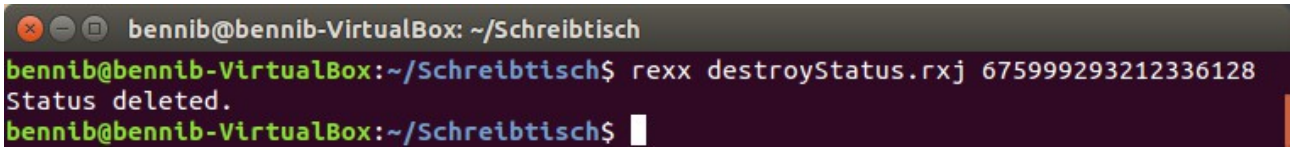

Figure 15: destroyStatus Terminal

The entire line of code would look like the following:

```
tw~destroyStatus(675999293212336128)
```

The next line prints the text to the terminal that the status has been deleted.

6.7. sendMessage

The sixth example demonstrates the sending of direct messages in twitter using the "sendDirectMessage()" method from the twitter4j library. Because there are now two inputs necessary, which are the receiver name and the text of the message itself, the program is build with the "PARSE PULL" command to make it simpler for the user to type the message. "PARSE PULL" waits for the input the user is making on the terminal and stores it into the variable that follows the command. In the example that is "Name" and "Nachricht". After receiving these two values we call the factory instance "tw" followed by a "~" and the method name "sendDirectMessage". Then inside the parenthesis after the method name the two variables "Name" and "Nachricht" are written, separated by a comma. The first variable has to be the receiver, the second the message body.

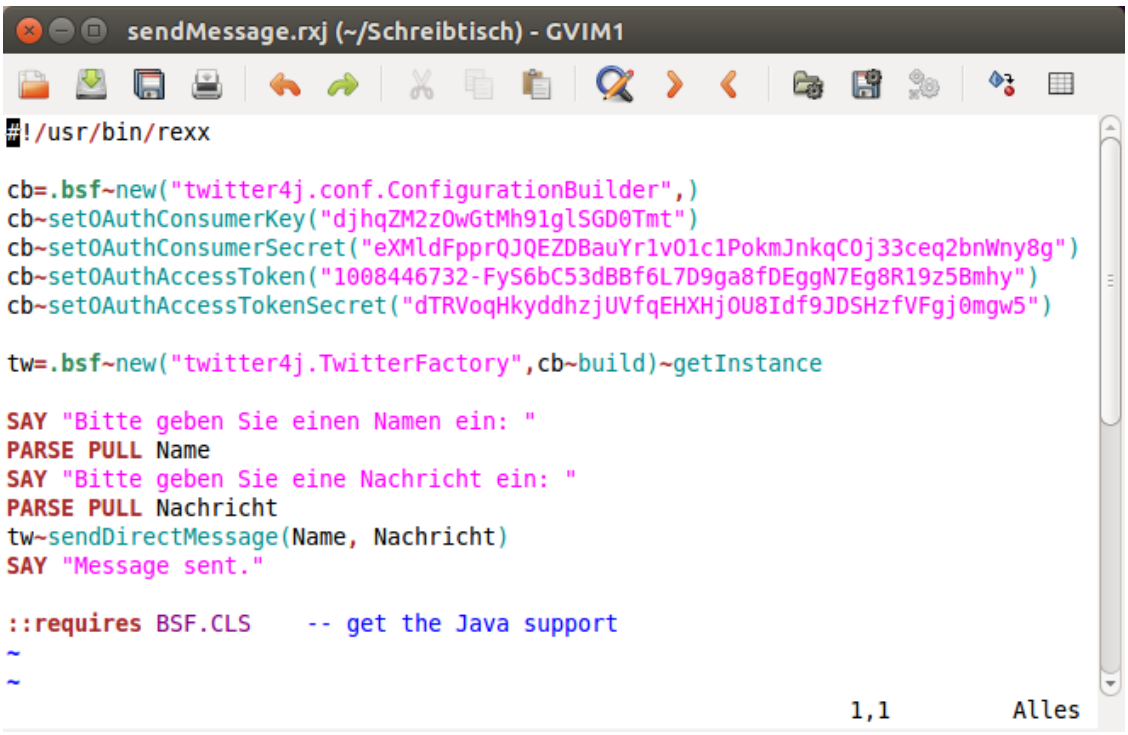

Figure 16: sendMessage

The last line is again a simple "SAY" statement printing "Message sent." to the

screen. The lines of code are displayed in Figure 16.

When executing the program first the user will be asked for a name he wants to send the message to. This is shown in Figure 17.

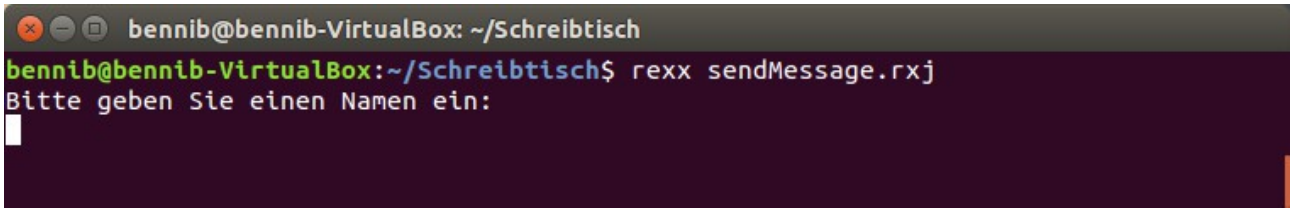

Figure 17: sendMessage Terminal 1

Then after entering a name the user will be asked to enter the message he wants to send to the receiver, shown in Figure 18.

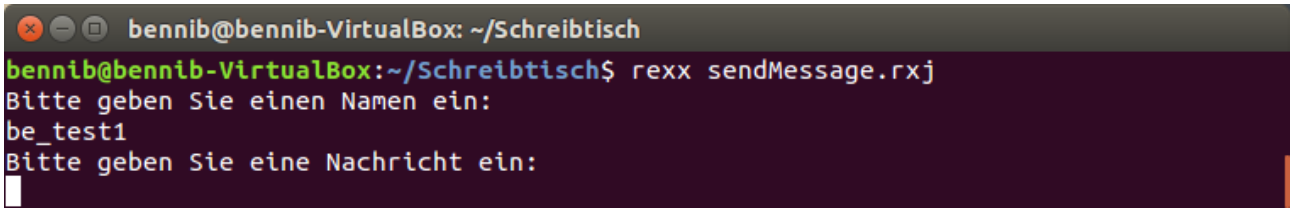

Figure 18: sendMessage Terminal 2

After entering a message, the message will be sent and the terminal shows the text "Message sent.", displayed in Figure 19.

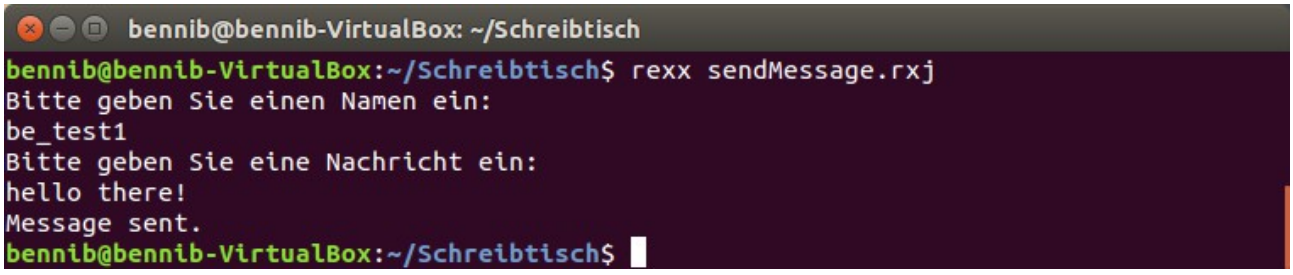

Figure 19: sendMessage Terminal 3

The entire line of code would look like the following:

tw~sendDirectMessage("be_test1", "hello there!")

In twitter itself the message is then displayed under "Messages" at the users account, shown in Figure 20.

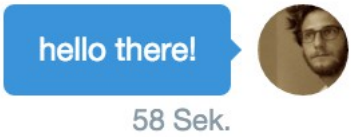

Figure 20: sendMessage Twitter

6.8. sentMessages

After sending a message in the sixth example, the seventh program displays a list of the sent Messages from the twitter account. The method used is "getSentDirectMessages" from the twitter4j library. As always we call the method with the factory instance "tw" followed by a "~" and the method name "getSentDirectMessages". The direct messages are then stored into the variable "messages". We need to put the messages into an array to create a list. Therefore we have to store the messages into an array with the command "messages~toArray" and save it into the "dir" variable. All the code is displayed in Figure 21.

sentMessages.rxj (~/Schreibtisch) - GVIM1

```
#!/usr/bin/rexx

cb=.bsf~new("twitter4j.conf.ConfigurationBuilder")
cb~setOAuthConsumerKey("djhqZM2zOwGtMh91glSGD0Tmt")
cb~setOAuthConsumerSecret("eXMldFpprQJQEZDBauYr1vO1c1PokmJnkqCOj33ceq2bnWny8g")
cb~setOAuthAccessToken("1008446732-FyS6bC53dBBf6L7D9ga8fDEggN7Eg8R19z5Bmhy")
cb~setOAuthAccessTokenSecret("dTRVoqHkyddhzjUVfqEHXHjOU8Idf9JDSHzfVFgj0mgw5")

tw=.bsf~new("twitter4j.TwitterFactory",cb~build)~getInstance

messages=tw~getSentDirectMessages
dir=.directory~new
dir=messages~toArray
SAY "Sent Messages:"
do i over dir
SAY i~getRecipientScreenName":" i~getText
END

::requires BSF.CLS    -- get the Java support
~
                                                    1,1           Alles
```

Figure 21: sentMessages

The next line prints the text "Sent Messages:" to the terminal screen.

Finally, the "DO OVER" loop stores every value of the "dir" array one by one separately into the variable "i" and prints it out in the next line. With the "i~getRecipientScreenName" we call the name of the receiver of the message and "i~getText" returns the text body from the message. The "END" statement ends the loop and the program is finished.

After executing the program through entering the line "rexx sentMessages.rxj" into the terminal, the sent Messages are displayed in the terminal as a list. The result is shown in Figure 22.

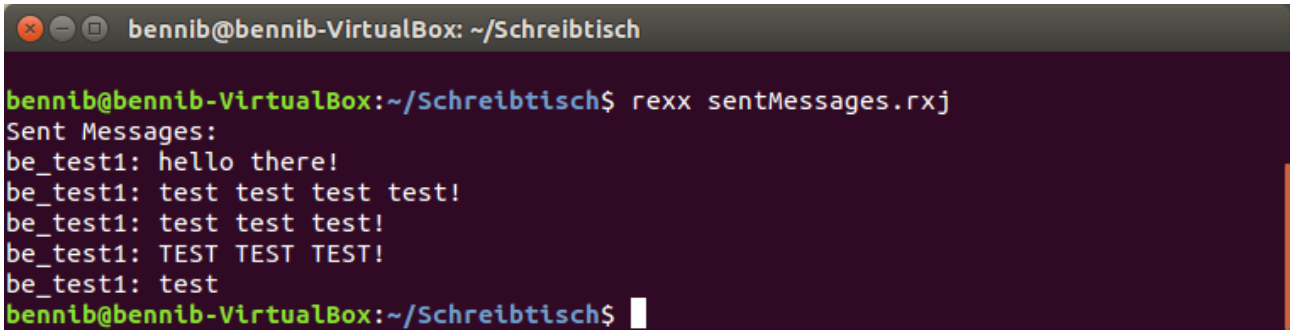

Figure 22: sentMessages Terminal

In twitter itself the sent Messages are displayed under Messages in the user account, shown in Figure 23.

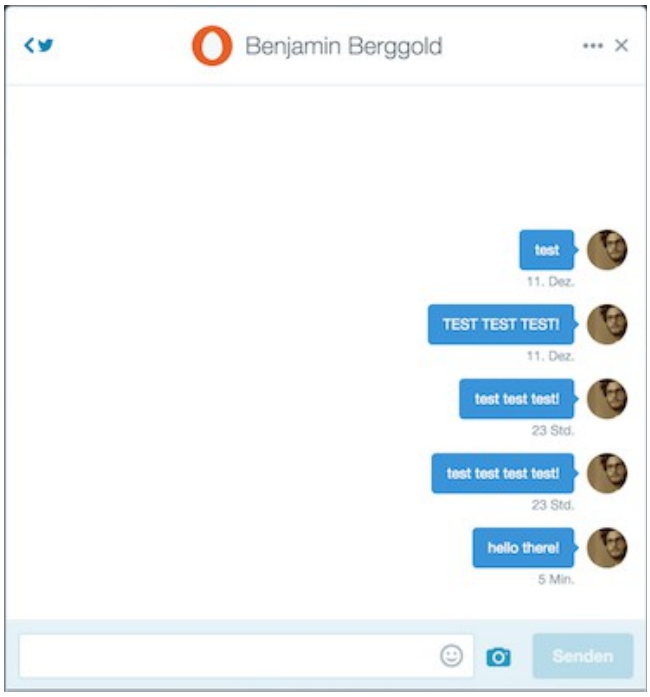

Figure 23: sentMessages Twitter

6.9. destroyFriendship

The eighth example demonstrates how to unfriend or unfollow a person on twitter. The method needed is "destroyFriendship()" from the twitter4j library. As always we call the method with the factory instance "tw" followed by a "~" and the method name "destroyFriendship". Inside the parenthesis, after the method name, the name of the user that we want to unfriend is entered. In this case we use the "arg" array again, as seen in Figure 24, which means it takes the input written after the program file at the execution in the terminal. The value is stored into the first position of the array ("arg(1)") and then handed inside the parenthesis of the method "destroyFriendship()".

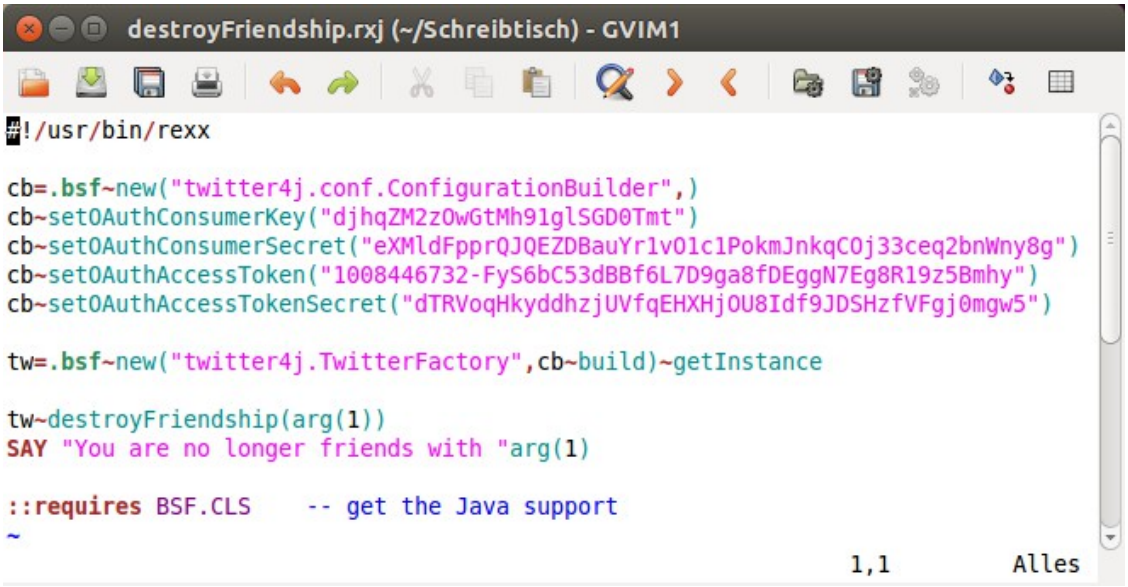

Figure 24: destroyFriendship

The execution of the program is displayed in Figure 25.

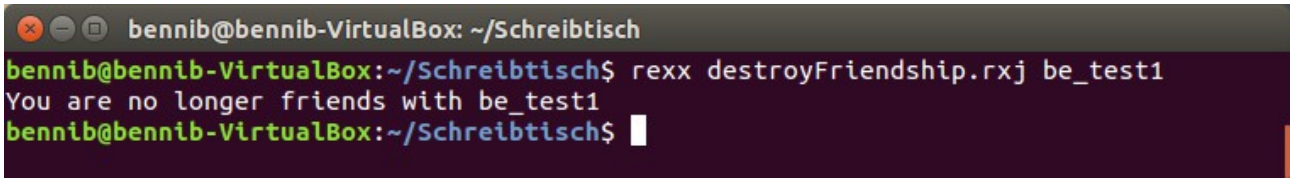

Figure 25: destroyFriendship Terminal

In this case "be_test1" is stored in the first position of the array "arg". The entire line of code would look like the following:

tw~destroyFriendship("be_test1")

The next line prints the text "You are no longer friends with " plus the name stored in "arg(1)" to the terminal. In this case "be_test1".

Finally, the result in twitter itself can be seen in Figure 26. We are no longer following "be_test1".

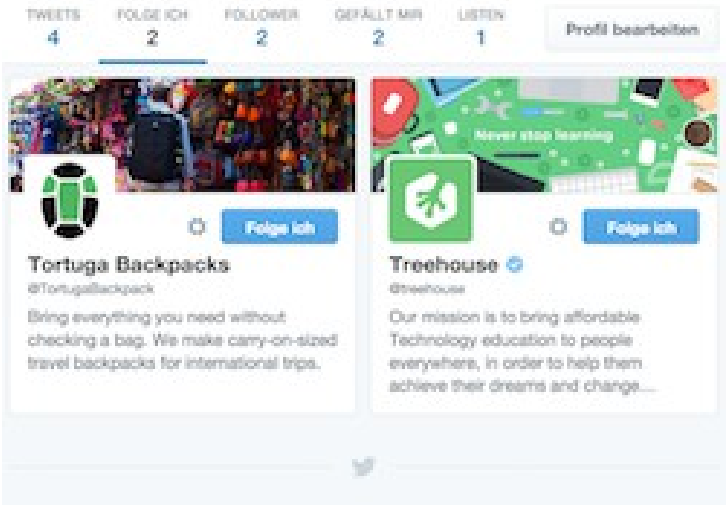

Figure 26: destroyFriendship Twitter

6.10. createFriendship

In the last example "be_test1" got unfollowed. The ninth example demonstrates how to follow him again. The method used is "createFriendship()" which is part of the twitter4j library. We call the method with the factory instance "tw" followed by a "~" and the method name

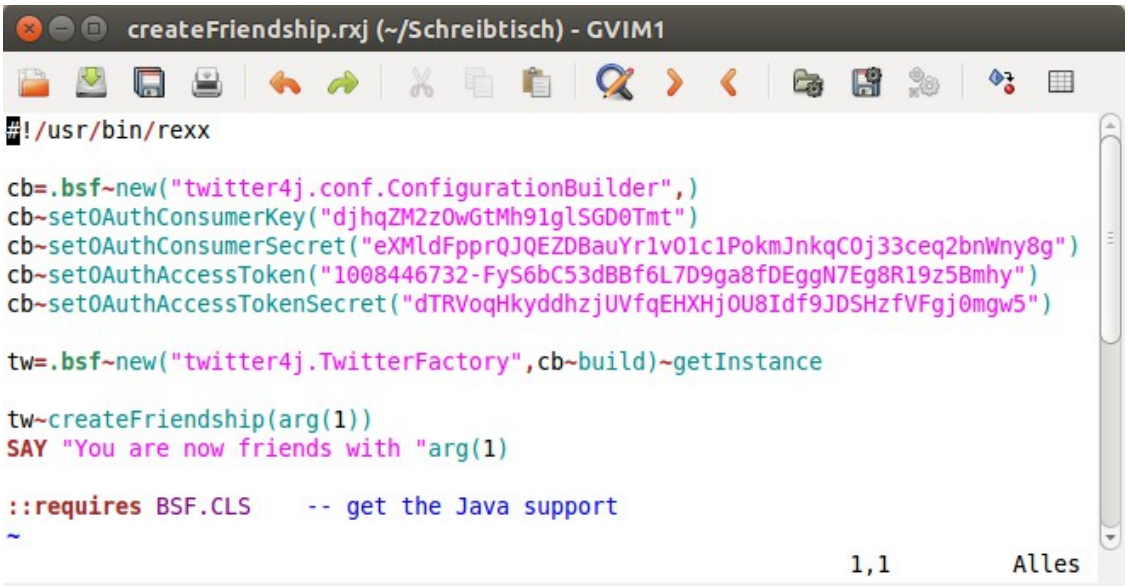

Figure 27: createFriendship

"createFriendship". Inside the parenthesis, after the method name, the name of the user we want to follow is entered. In this case we use the "arg" array again, as seen in Figure 27, which means it takes the input written after the program file at the execution in the terminal. The value is stored into the first position of the array ("arg(1)") and then handed inside the parenthesis of the method "createFriendship()".

The execution of the program is shown in Figure 28.

```
bennib@bennib-VirtualBox: ~/Schreibtisch
bennib@bennib-VirtualBox:~/Schreibtisch$ rexx createFriendship.rxj be_test1
You are now friends with be_test1
bennib@bennib-VirtualBox:~/Schreibtisch$
```

Figure 28: createFriendship Terminal

In this case "be_test1" is stored in the first position of the array "arg". The entire line of code would look like the following:

tw~createFriendship("be_test1")

The next line prints the text "You are now friends with " plus the name stored in "arg(1)" to the terminal. In this case "be_test1".

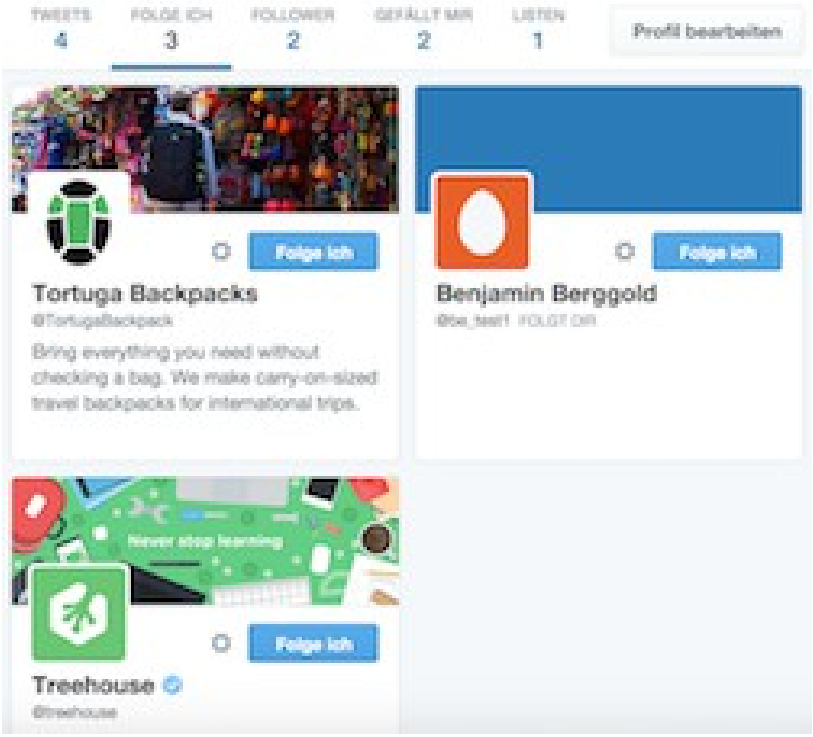

Figure 29: createFriendship Twitter

Finally, the result in twitter itself can be seen in Figure 29.

6.11. getTrends

The tenth and last example displays the different trends locations in a list together with their WOEID (Where On Earth IDentifier). The method used is "getAvailableTrends" from the twitter4j library. As always we call the method with the factory instance "tw" followed by "~" and the method name "getAvailableTrends". The trends are then stored into the variable "trends". We need to put the trends into an array to create a list. We can store the trends into an array with the command "trends~toArray" and save it into the "dir" variable. All the code is displayed in Figure 30.

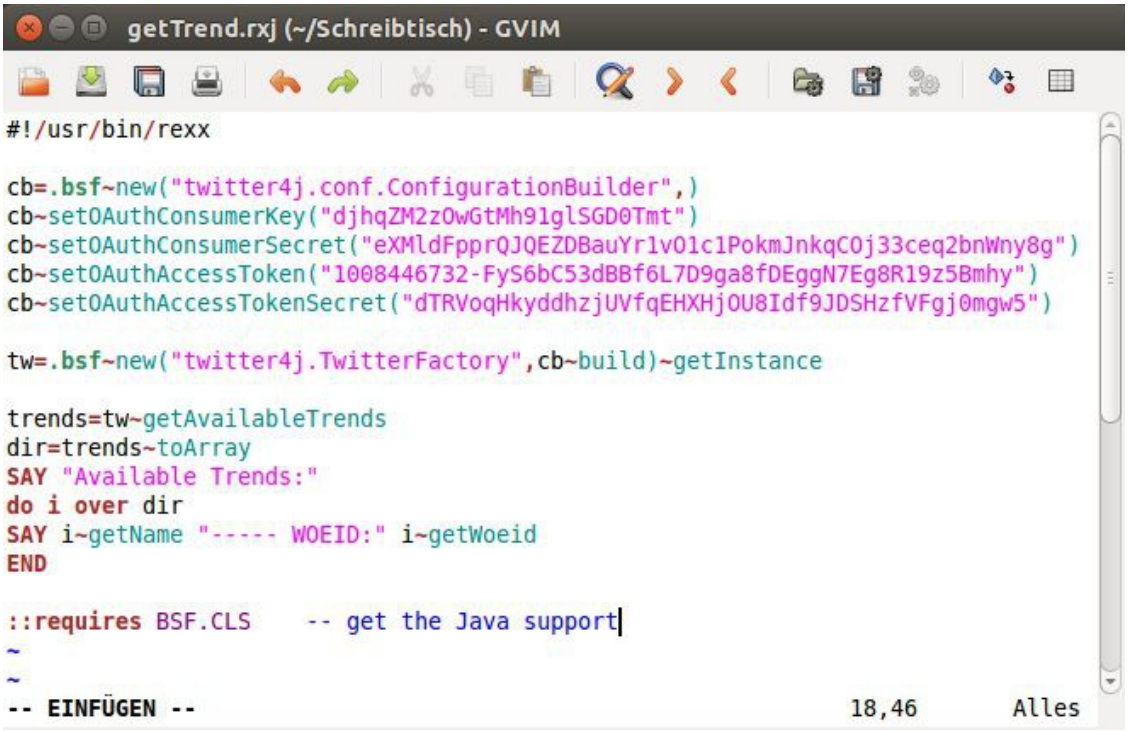

Figure 30: getTrends

The next line prints the text "Available Trends:" to the terminal screen.

Finally, the "DO OVER" loop stores every value of the "dir" array one by one separately into the variable "i" and prints it out in the next line. With the "i~getName" we call the name of the country and "i~getWoeid" returns the corresponding WOEID. The "END" statement ends the loop and the program is

finished.

After executing the program through entering the line “rexx getTrends.rxj” into the terminal, the trends locations are displayed in the terminal as a list. Part of the result is shown in Figure 31.

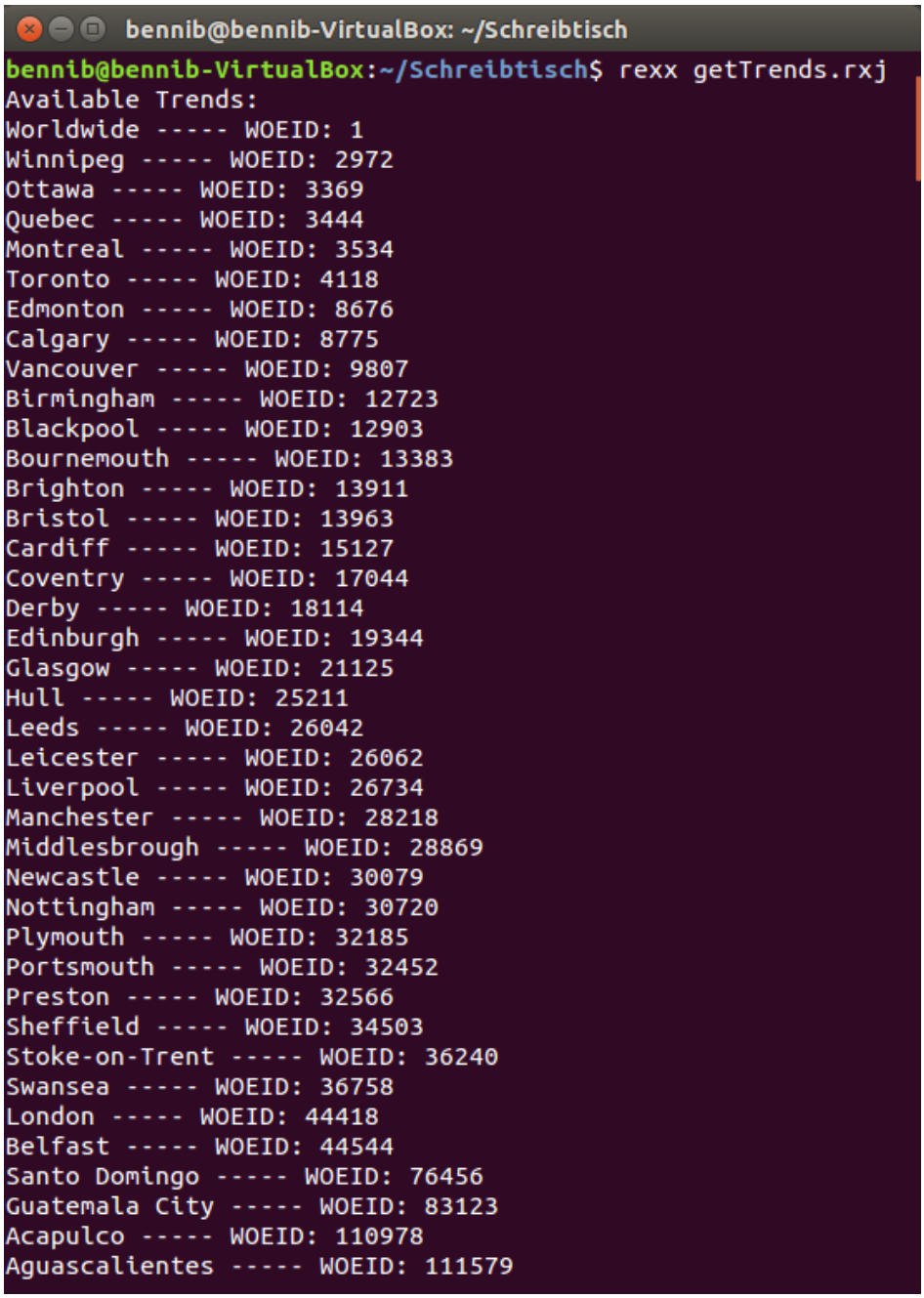

Figure 31: getTrends Terminal

In twitter itself the trends are displayed on the left of the home screen, shown in Figure 32.

Figure 32: getTrends Twitter

7. Conclusion

In conclusion, the effort to communicate with the Twitter API through BSF4ooRexx is fairly small. Using the Twitter4J library with its preconstructed methods makes it easy to access the different functions the API has to offer. With BSF4ooRexx it is possible to use the Java methods with the Rexx programming language and write working programs with only few lines of code. Twitter4J has extensive documentation on its homepage which makes it easy to read and learn the methods and their functionality. OoRexx is a quickly learnable and simple programming language and with the BSF4ooRexx extension all Java methods can be used in Rexx. Therefore there is no downside in using ooRexx making it perfect for programming the Twitter API.

Bibliography

1: Wikipedia: Twitter, visited 12/2015
https://en.wikipedia.org/w/index.php?title=Twitter&oldid=694934957

2: Techterms: Tweet Definition, visited 12/2015
http://techterms.com/definition/tweet

3: Wikipedia: Application Programming Interface, visited 12/2015
https://en.wikipedia.org/w/index.php?
title=Application_programming_interface&oldid=694622708

4: Wikipedia: Representational State Transfer, visited 12/2015
https://en.wikipedia.org/w/index.php?
title=Representational_state_transfer&oldid=694487867

5: Twitter for Developers: Rate Limiting, visited 12/2015
https://dev.twitter.com/rest/public/rate-limiting

6: Twitter for Developers: OAuth, visited 12/2015
https://dev.twitter.com/oauth

7: Wikipedia: Java, visited 12/2015
https://en.wikipedia.org/w/index.php?
title=Java_(programming_language)&oldid=695946012

8: The Rexx Language Association, visited 12/2015
http://www.rexxla.org/rexxlang/

9: Rony G. Flatscher, Introduction to Rexx and ooRexx, 2013

10: Twitter4J Library Homepage, visited 12/2015
http://twitter4j.org/en/index.html