Vienna University of Economics and Business

Institute for Management Information Systems

Course 0208, Projektseminar aus Wirtschaftsinformatik (Schiseminar)

Rexxoid: Running Rexx on Android Systems

Julian Reindorf, 1151548

Supervisor: ao. Univ.Prof. Mag. Dr. Rony G. Flatscher

December 25, 2014

Declaration of Authorship

"I do solemnly declare that I have written the presented research thesis 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Where I have used thoughts from external sources, directly or indirectly, published or unpublished, this is always clearly attributed.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."

Date:

Signature:

Contents

List of Figures

List of Tables

Listings

Abstract

Rexx runs on several operating systems, including Windows, MacOS and Linux. However, the recent development towards mobile devices leads to the need of running Rexx also on the most wide-spread mobile device operating system: Android. The Rexxoid interpreter enables Android devices to execute Rexx scripts. Furthermore commands can be sent to the Android shell. This paper describes Rexxoid and presents several short examples. In addition, differences to another approach of running Rexx on Android (BRexx) are discussed.

1 Introduction

This section briefly introduces Android, Rexx and Rexxoid. The afterwards following sections will discuss:

- The Rexxoid application (section 2)
- Rexxoid nutshell examples (section 3)
- Limitations of Rexxoid (section 4)
- Comparison of Rexxoid and BRexx (section 5)

1.1 Android

Android, an operating system (OS) for mobile devices, was first announced in 2007. It is developed by the Open Handset Alliance, which is a multinational team of IT companies, founded by Google Inc [Open14]. The success story of this OS is amazing. Android showed an incredible growth within the quarters between Q1 2009 and Q3 2010. In this timeframe, sales grew from 0 (Q1 2009) to more than 8 million (Q3 2010). By this it outran sales of Blackberry and Apples iOS, thereby becoming the first real threat for the still growing iPhones sales. [see Butl11, 4]

Recent statistics show that Android reached a market share of about 85% in Q3 2014. iOS lags far behind with its approximately 12%. Other competitors struggle hard to stay in the market, sharing the last few percent. [see Inte14]

Android is running on a modified Linux kernel and comes with many built-in activities. Its store is free, in contrast to Apples app store. As a result, applications can be published easily. Using Android, users can control the security settings on their own. Applications need permission to use services of the device. [see Butl11, 5]

Smartphones have drastically changed the way people use their phones. While, in earlier times, phones were used for calls and sending sms only, now a smartphone is an allround device, assisting people in all areas of their lives. Android heavily contributed to this change. [see Butl11, 5]

1.2 Rexx

The programming language Restructured Extended Executor (Rexx) was initiated in 1979 by Mike F. Cowlishaw to provide an easier language for IBM mainframes than Exec 2. The easy to understand language was developed further and in 2005 Open Object Rexx (ooRexx) was published by the Rexx Language Association. [see Flat13, iii f.]

The main features of ooRexx are: [see Flat13, iv ff.]

- It is "backward compatible". Therefore, normal Rexx scripts run unchanged under ooRexx.
- As its name says, it is object oriented. Many useful classes are included out of the box.
- Multithreading is possible.
- It is a fast interpreter.
- ooRexx comes with a comprehensive and detailed documentation with a lot of examples.
- It is free and open source.
- It runs unchanged on 32 and 64 bit operating systems.

The Bean Scripting Framework for ooRexx (BSF4ooRexx) package provides the possibility to camouflage Java objects as Rexx objects. This package increases the possible number of applications for ooRexx even further. [see Flat13, 159 f.]

2 Rexxoid

Rexxoid, a Rexx interpreter, was published by Pierre Richard [see Goog14]. It comes with an Android application which provides a simple text editor and the possibility to execute Rexx code. Additionally, commands can be sent to the Android shell. This enables the user to start activities and therefore build scripts to automate processes on Android devices. Unfortunately, there is no documentation for Rexxoid and the number of users working with it seems to be very low.

2.1 Getting Started

The Rexxoid apk file can be downloaded to one's Android device from the Google Play Store, it is named "Rexx for Android", developed by Jaxo, Inc. After installation, there are a few example scripts that can be executed (see Figure 1). Most of them include only Rexx code and do not send commands to Android itself. One example is more interesting: the "Android System Test" (see Figure 2a and Figure 2b for the editor and execution view of the script). It uses the application manager and opens the settings GUI. If errors arise when executing the script, they will be displayed in the output field.

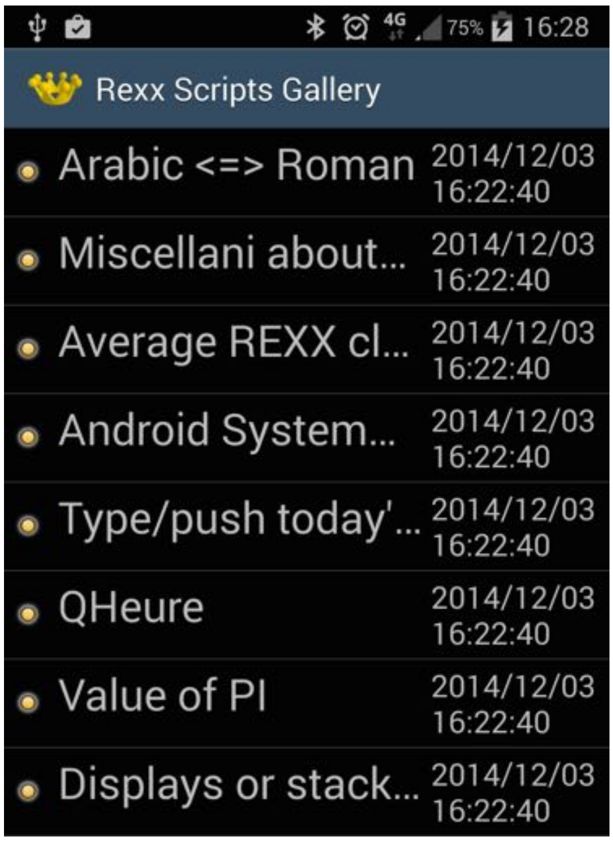
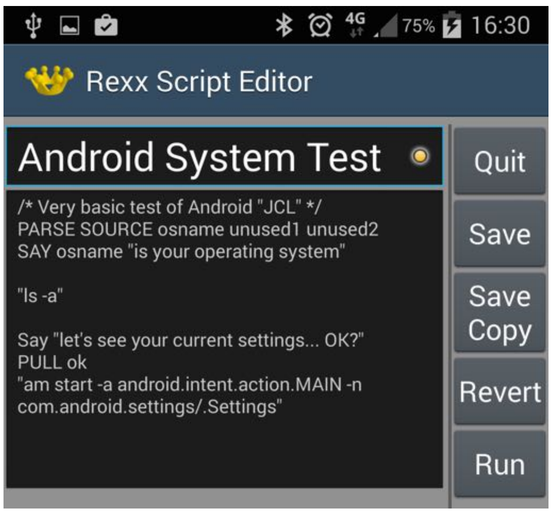
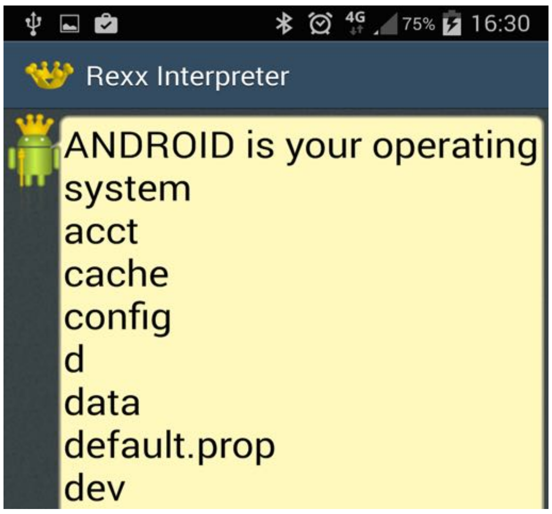

16:28
Rexx Scripts Gallery
Arabic <=> Roman 2014/12/03 16:22:40
Miscellani about... 2014/12/03 16:22:40
Average REXX cl... 2014/12/03 16:22:40
Android System... 2014/12/03 16:22:40
Type/push today'... 2014/12/03 16:22:40
QHeure 2014/12/03 16:22:40
Value of PI 2014/12/03 16:22:40
Displays or stack... 2014/12/03 16:22:40

Figure 1: The Rexxoid menu



```
/* Very basic test of Android "JCL" */
PARSE SOURCE osname unused1 unused2
SAY osname "is your operating system"

"ls -a"

Say "let's see your current settings... OK?"
PULL ok
"am start -a android.intent.action.MAIN -n
com.android.settings/.Settings"
```

(a) Script editor



```
ANDROID is your operating
system
acct
cache
config
d
data
default.prop
dev
```

(b) Output console

Figure 2: Rexxoid screens

2.2 Permissions

For the execution of some commands, the Rexxoid application needs additional permissions. Section 2.2.1 describes how to grant additional permissions, while section 2.2.2 discusses a frequently thrown permission exception.

2.2.1 Adding New Permissions

Like every other Android application, Rexxoid has a Manifest file where every used permission is stated explicitly. Therefore, every script that runs in Rexxoid has the same permissions as the application itself. By default, only two permissions are used:

- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_EXTERNAL_STORAGE

That means, if a script calls an intent (or sends any other command to the Android shell) where additional permissions are needed, this will rise a permission denied exception. In the Android operating system an intent is a messaging object, which allows to trigger specific functions of applications [see Wiki14]. Therefore, intents can e.g. be used in Rexxoid scripts to launch other applications.

To modify the permissions the Rexxoid project needs to be downloaded to a computer, its Manifest file needs to be modified and then it must be reinstalled on the mobile device.

To do so, the Rexxoid project needs to be downloaded (e.g. from GitHub: `https://github.com/Jaxo/yaxx/tree/master/android`) first. Next, the downloaded project has to be opened in a suitable IDE (Integrated development environment) for Android programming, e.g. Eclipse with the ADT (Android Development Tools) plugin installed. After that, the AndroidManifest.xml file has to be edited. For each desired permission one line needs to be added (e.g. `<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />` for the permission to create alarms). After all desired permissions are set, the device (i.e. smartphone or tablet) must be connected with a USB cable to the computer. USB debugging (an option in the device's developer options) has to be enabled on the target device. Now, the modified Rexxoid application can be reinstalled via the USB cable from the computer.

If there is no effect, one should try to uninstall the Rexxoid application first, and then install it via Eclipse again. Either way, all scripts should be kept in a backup, because reinstalling the application will delete all of them. In the Rexxoid project, there is a folder "assets", which includes the folder "rexx". Any script in this folder will be available on the device after installation. Note that the name of a script stored in this directory is irrelevant, however, the first line must be a comment, which will then (on the device) be displayed as the scripts name.

2.2.2 Permission: INTERACT_ACROSS_USERS_FULL

Many commands lead to the following error:

```
java.lang.SecurityException: Permission Denial: startActivity asks to run as
 user -2 but is calling from user 0; this requires
android.permission.INTERACT_ACROSS_USERS_FULL
```

Although the error message says so, adding the `android.permission.INTERACT_ACROSS_USERS_FULL` permission to the Android Manifest does not help in this case. Instead, adding the optional parameter `--user 0` to the executed command fixes the problem.

2.3 Debugging

If errors occur, they will be displayed in the output area. Since these are mostly very long messages, it can be difficult to read them. However, the log can be viewed from a computer as well. For doing so, the device and the computer have to be connected with a USB cable. Furthermore, USB debugging needs to be enabled. Then, if not already done, the path to the adb (Android debug bridge), which comes with the installation of the Android Software Development Kit [see Andr14b], has to be added to the $PATH variable. After that, the terminal can be used to read the logcat by typing `adb logcat` in it. This command will lead to a lot of messages (not only logged by Rexxoid). To filter messages, type e.g. `adb logcat | grep -i "rexx"` (this will display only messages that include "rexx", case insensitive). These steps apply to a Mac OS environment and may differ on other operating systems.

2.4 Shell Commands

Any code enclosed by quotation marks (e.g. "code") that is not an argument to a Rexx function, will be sent to the Android shell.

Commands using the application manager ("am") follow this pattern:

```
am <command> <mandatory arguments> <optional extra values>
```

- "am" stands for application manager.
- "<command>" is to be replaced by the desired command.
- "<mandatory arguments>" is to be replaced by e.g. the intent to be started.
- "<optional extra values>" can be left empty or be replaced by specific extra values.

One important command is the "start" command. It is used to start an activity and follows this pattern:

```
am start -a <full qualified intent name> <optional extra values>
```

- "start" is the command to be executed.
- "-a <full qualified intent name>" specifies the intent action.

For a list of all available commands, please see
`http://developer.android.com/tools/help/adb.html` [see Andr14a].

2.5 Shell Extra Values

Most commands accept additional parameters which have to be defined by their datatype. These parameters follow the pattern:

```
<option> <key> <value>
```

See Table 1 for some of the most important datatypes.

Description	Option	Example
String	--es	--es <key> 'hello World'
Boolean	--ez	--ez <key> true
Integer	--ei	--ei <key> 5

Table 1: Datatypes for intents

For a list of all available extra values that can be added, see `http://developer.android.com/tools/help/adb.html` [see Andr14a].

3 Nutshell Examples

In this section nutshell examples, using the concepts described above, will be shown and described.

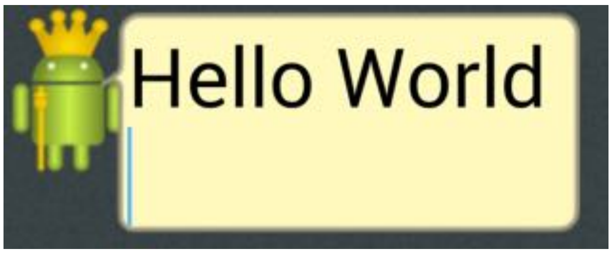
3.1 Hello World

The first example traditionally is a program that simply prints the string "Hello World". To follow this intention, here is Hello World in Rexxoid:

```
SAY "Hello World"
```

Listing 1: Hello World

As can be seen, this short program runs unchanged on a desktop computer and on an Android device running Rexxoid. Figure 3 shows the output of the program.



Hello World

Figure 3: Hello World example

3.2 Setting Alerts

The following example asks the user for some settings and subsequently creates a corresponding alert.

This example requires the following permission (see section 2.2 for more information): `com.android.alarm.permission.SET_ALARM`.

```
 1 SAY "*Alert Creation*"
 2 SAY "Hour?"
 3 PULL hour
 4 SAY "Minute?"
 5 PULL minute
 6 SAY "Message?"
 7 PULL message
 8 "am start -a android.intent.action.SET_ALARM --user 0 "     ,
 9     "--ei android.intent.extra.alarm.HOUR " hour            ,
10     "--ei android.intent.extra.alarm.MINUTES " minute       ,
11     " -e  android.intent.extra.alarm.MESSAGE '" message "'" ,
12     "--ez android.intent.extra.alarm.VIBRATE false "        ,
13     "--ez android.intent.extra.alarm.SKIP_UI true"
14 SAY "Done!"
```

Listing 2: Create an alert

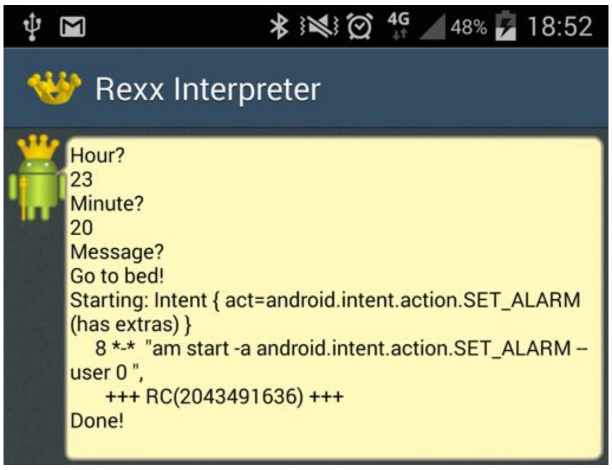
Line 1 informs the user about the purpose of the script. Lines 2 to 7 are simple Rexx code that ask the user for some settings. The next command (lines 8 to 13) is sent to the Android shell. The last one informs the user that the script has finished.

Line 8 tells the activity manager (am) to start an activity. There are some additional options which are described below.

- **-a android.intent.action.SET_ALARM**: Specifies the action. This needs to be the fully qualified intent name.
- **--user 0**: Specifies the user (Without this, there will be an exception).
- **--ei android.intent.extra.alarm.HOUR hour**: Sets the alarm hour to the value of `hour`. Must be an integer value.
- **--ei android.intent.extra.alarm.MINUTES minute**: Sets the alarm minutes to the value of `minute`. Must be an integer value.
- **-e android.intent.extra.alarm.MESSAGE message** : Sets the alarm message to the value of `message`. Must be a quoted string value.
- **--ez android.intent.extra.alarm.VIBRATE false**: Specifies that the alarm plays a sound and that the device does not vibrate. If set to true, there will be no audio notification, instead the device only vibrates.

- **--ez android.intent.extra.alarm.SKIP_UI true**: Defines that the alarm user interface should not be shown. If omitted or set to false, the alarm UI will be opened after alarm creation.

Figure 4 shows the script in action. The user wants to create an alert at 23:20 with the message "Go to bed!". As can be seen, Rexxoid prints system messages on activity start. Whenever an activity is started, a TRACE-statement, which contains the return code of Android, is printed (here, "+++ RC(2043491636) +++"). The script ends with the "Done!" statement.



```
Hour?
23
Minute?
20
Message?
Go to bed!
Starting: Intent { act=android.intent.action.SET_ALARM
(has extras) }
    8 *-* "am start -a android.intent.action.SET_ALARM --
user 0 ",
      +++ RC(2043491636) +++
Done!
```

Figure 4: Alert creation - user interaction

After the alert is created, one can go to the alert menu and the newly created alert is displayed as in Figure 5.

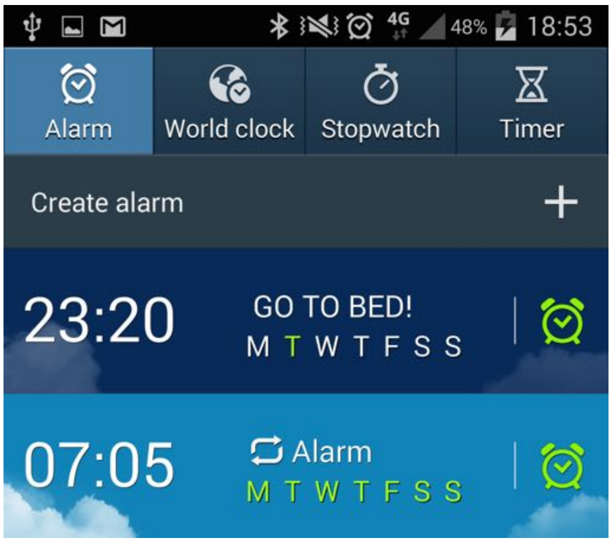


Figure 5: Alert created

3.3 Adding Calendar Events

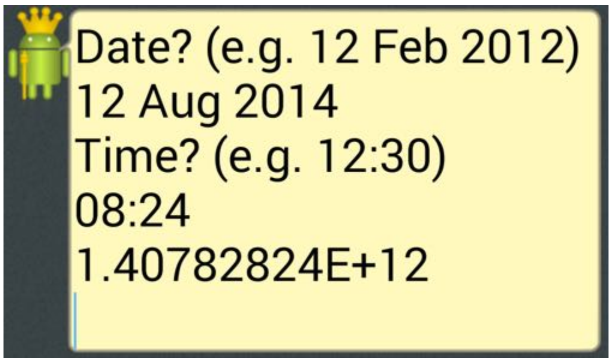
This example illustrates how to create a simple new event for the calendar. The script does not require any special permissions.

```
NUMERIC DIGITS 18

SAY "Title?"
PULL title

SAY "Date? (e.g. 12 Feb 2012)"
PARSE PULL inputDate

SAY "Time? (e.g. 12:30)"
PULL inputTime
PARSE VAR inputTime inputHours ":" inputMinutes

SAY "Duration? (in hours)"
PULL durationHours

daysSinceEpoch = DATE("B", inputDate) - DATE("B", "01 Jan 1970")
millisSinceEpoch = daysSinceEpoch*24*60*60*1000

millisSinceEpoch = millisSinceEpoch + (inputHours*60+inputMinutes)*60*1000

millisSinceEpoch = millisSinceEpoch - 60*60*1000

"am start -a android.intent.action.INSERT --user 0 "          ,
    "-d Events.CONTENT_URI -t vnd.android.cursor.dir/event " ,
    "--es title '" title "'"                                 ,
    "--el beginTime " millisSinceEpoch                       ,
    "--el endTime " millisSinceEpoch+durationHours*60*60*1000
```

Listing 3: Create calendar event

To create events, the begin- and endtime must be provided in milliseconds since the Linux epoch (i.e. since the beginning of the Unix time, 01. January 1970). Therefore, some calculations have to be done before the activity to actually create the event can be started.

Line 1 tells the compiler to do calculations with a precision of 18 digits. As the default precision of 9 digits is insufficient (the numbers for the milliseconds can exceed 10^{12}, i.e. 13 digits), the used precision has to be increased. Figure 6 shows the milliseconds output without increased precision. The activity, which is started later, does not accept this notation. Line 3 to 14 ask the user for a title, the start date and time and the duration of the event. `PARSE PULL` is used to keep the input date case-sensitive (otherwise Rexx would transform the input to uppercase characters).



```
Date? (e.g. 12 Feb 2012)
12 Aug 2014
Time? (e.g. 12:30)
08:24
1.40782824E+12
```

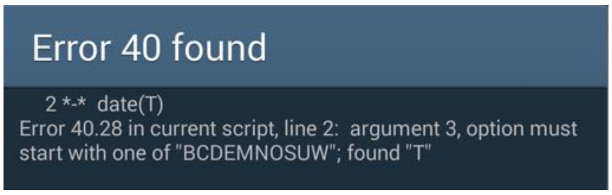
Figure 6: Calculating milliseconds without specified precision

After that, the milliseconds since the Linux epoch are calculated. Line 21 corrects the time difference between GMT and Austrian wintertime. For the sake of simplicity summertime is ignored, but therefore events created in the summertime span will have a one hour difference to the actual desired time.

Line 23 starts the activity and some additional parameters are defined:

- **-a android.intent.action.INSERT**: Defines the action to be executed.
- **--user 0**: Specifies the user (Without this, there will be an exception).
- **-d Events.CONTENT_URI**: Specifies the intent's data URI.
- **-t vnd.android.cursor.dir/event**: Specifies the intent's MIME type.
- **--es title title** : Specifies the title for the event. Here, the variable `title`, which holds the user's input, is used.
- **--el beginTime millisSinceEpoch** : Defines the start time for the event. Must be given in milliseconds since epoch (since 01. January 1970 00:00:00).
- **--el endTime millisSinceEpoch**: Defines the end time for the event. Must be given in milliseconds since epoch (since 01. January 1970 00:00:00).

Unfortunately, `Date("F")` and `Date("T")` do not work in Rexxoid, so milliseconds have to be calculated in a little more complex way, as in the example above. Figure 7 shows the error message that comes up if `Date("T")` is called.

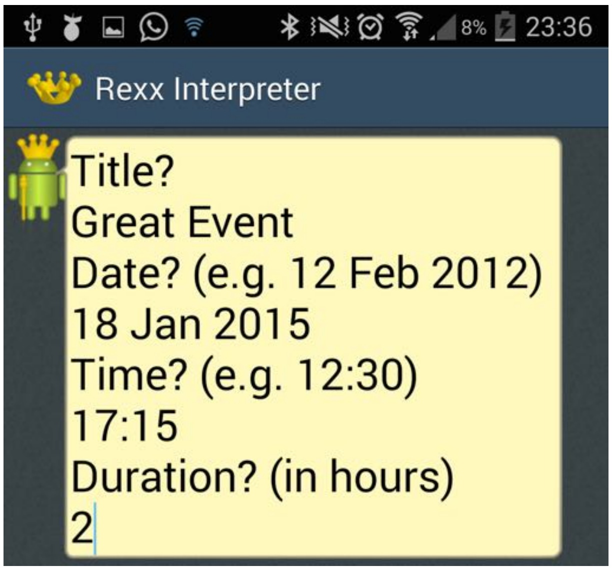


Error 40 found

```
  2 *-* date(T)
Error 40.28 in current script, line 2: argument 3, option must
start with one of "BCDEMNOSUW"; found "T"
```

Figure 7: The date(T) error

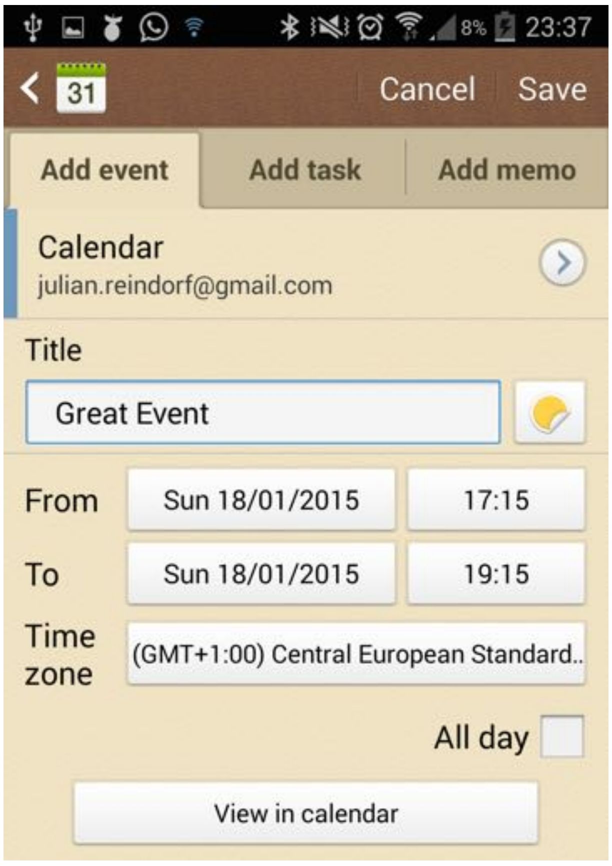
Figure 8 shows the result of this example. The user wants to create a two hour long event on January 18, 2015 17:15 with the title "Great Event".



```
Title?
Great Event
Date? (e.g. 12 Feb 2012)
18 Jan 2015
Time? (e.g. 12:30)
17:15
Duration? (in hours)
2
```

Figure 8: Event creation - user interaction

After the user enters the duration as shown in Figure 8, the calendar event creation screen (Figure 9) is displayed. After modifying the last settings, the event can finally be saved.



Calendar
julian.reindorf@gmail.com

Title: Great Event

From: Sun 18/01/2015 17:15

To: Sun 18/01/2015 19:15

Time zone: (GMT+1:00) Central European Standard..

All day ☐

Figure 9: Event creation - settings

3.4 Command Line

This example lets the user enter commands which are then sent to the Android shell. The user can stop the script by typing 'stop' or 'STOP'.

```
DO WHILE TRANSLATE(input) <> 'stop'
    PARSE PULL input
    input
END
SAY "*Exited*"
EXIT
```

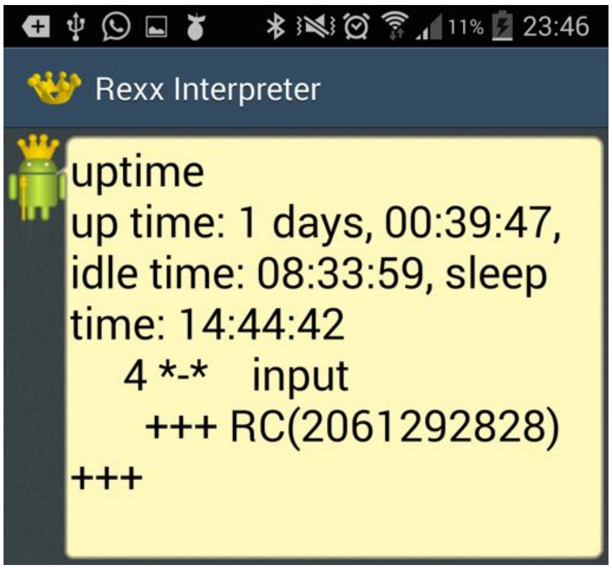
Listing 4: Send commands to Android shell

Some Linux commands that could be interesting to execute are:

- **ls**: Lists all files in the current directory.
- **ls -la**: Lists all files in the current directory (with more information than only `ls`).

- **ls /system/bin**: Shows all available shell programs. However, most of them require root user permissions to be executable.
- **uptime**: Shows the current uptime of the device.

In Figure 10 the user entered the "uptime" command. The command is forwarded to the Android shell and the result is displayed immediately.



```
uptime
up time: 1 days, 00:39:47,
idle time: 08:33:59, sleep
time: 14:44:42
     4 *-*    input
       +++ RC(2061292828)
+++
```

Figure 10: Command line showing uptime

3.5 Creating Emails

This example shows how to create an email with a specified receiver, subject and text message.

```
text = ""
DO i=1 TO 100
    text = text i
END
"am start -a android.intent.action.SEND --user 0 " ,
    "-t 'text/plain' "                              ,
    "-e to 'john.doe@gmail.com' "                   ,
    "-e android.intent.extra.SUBJECT '1 to 100!' " ,
    "-e android.intent.extra.TEXT '"text"'"
```

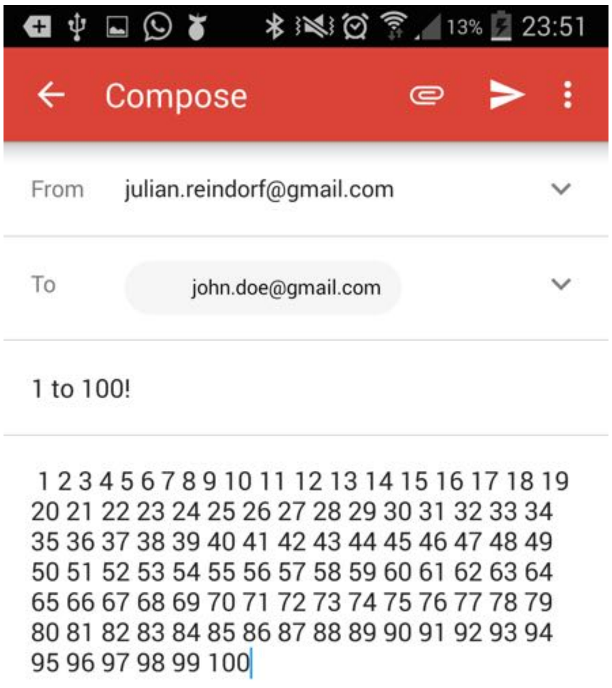
Listing 5: Create email

The purpose of lines 1-4 is to have some text in the variable `text`. The fifth line calls the action with the following settings:

- **-a android.intent.action.SEND**: Defines the action to be called.
- **--user 0**: Specifies the user (Without this, there will be an exception).
- **-t 'text/plain'**: Specifies the MIME-Type. This specifies it as plain text.
- **-e to 'john.doe@gmail.com'**: The extra string argument "to" specifies the receiver of the email.
- **-e android.intent.extra.SUBJECT '1 to 100!'**: Specifies "1 to 100!" to be the subject of the email.
- **-e android.intent.extra.TEXT text**: Specifies the value of the `text` variable to be the body of the email.

Note that the email is not sent by the script. It only opens the email application with preallocated fields.

This example includes no user interaction. Therefore, the result will always be the same, as shown in Figure 11. Note that the sender is not explicitly set in the script, instead the default sender is chosen.



From julian.reindorf@gmail.com

To john.doe@gmail.com

1 to 100!

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
95 96 97 98 99 100
```

Figure 11: Rexxoid created email

3.6 Opening Google Maps

The following script opens the Google Maps application showing the requested location. The location can be almost anything like a street, city, country, water, mountain etc.

```
1 SAY "What do you want to see?"
2 input = LINEIN()
3 location=""
4 DO WHILE LENGTH(input) > 0
5     PARSE VAR input char +1 input
6     location=location|| "%"||C2X(char)
7 END
8 "am start -a android.intent.action.VIEW --user 0 -d 'geo:0,0?q="location"'"
```

Listing 6: Open Google Maps

The user has to enter the requested location. After that, every character of the entered location string is converted to its hexadezimal value, preceded by a percentage sign ("%"). This step is necessary, otherwise characters except for letters and numbers will not work correctly (e.g. blank (" ") or ampersand ("&")).

Line 9 starts the activity with the following settings:

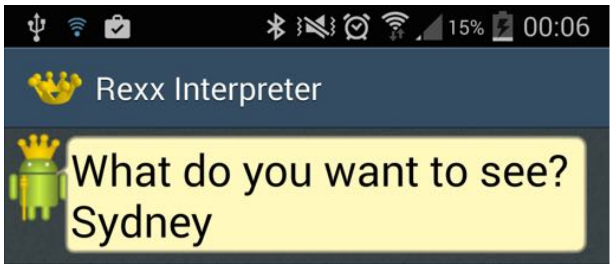
- **-a android.intent.action.VIEW**: Defines the action to be called.
- **--user 0**: Specifies the user (Without this, there will be an exception).
- **-d 'geo:0,0?q="location"'**: Specifies the intent's data URI. In this case, the value of the variable `location` is set as the "q" parameter. In this example, the latitude (0) and longitude (0) have no effect.

Alterations of the intent used above can be:

- **-d 'geo:50,-70'** will show the map at the specified latitude (50) and longitude (-70).
- **-d 'geo:50,-70?z=10'** will show the map at the specified coordinates with the specified zoom (here, 10). The zoom level can be an integer between 1 (maximal zoom out) and 20 (maximal zoom in).

Note that it is not possible to use the "q" and the "z" parameter in the same activity call.

Figure 12 shows the script in action. As can be seen, the user wants to see Sydney.



What do you want to see?
Sydney

Figure 12: User chooses location

After the user choses the desired location, Google Maps starts and focuses on it. This can be seen in Figure 13.

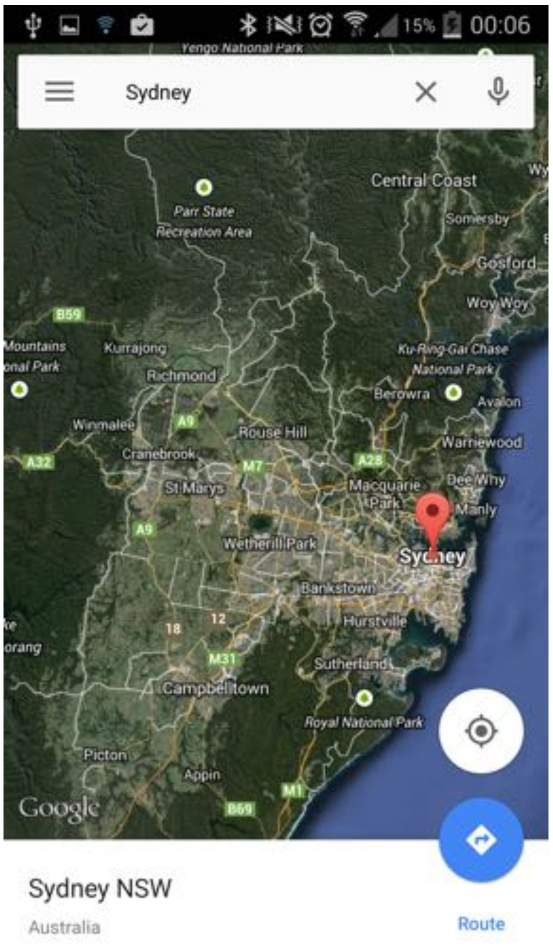


Figure 13: Google Maps showing Sydney

3.7 Using the Speaker

This short example shows how to use the speakers to read out text.

```
CALL CHAROUT "Speaker:lang=en", "I am the voice of Rexxoid"
CALL CHAROUT "Speaker:lang=de", "Und ich kann auch andere Sprachen"
CALL CHAROUT "Speaker:lang=fr", "Au revoir"
```

Listing 7: Use speaker

The example consists of only one command (which is executed a few times). The first argument is an URI and defines the language for the reading aloud. It follows the pattern "Speaker:lang=<Code>", where <Code> is to be replaced by the two character language abbreviation. The second argument defines the text to be read. Of course, it is also possible to specify a different language in the URI and the text to be actually read aloud (e.g. "de" for German reading, and supplying an English text). This will, however, result in a hard to understand sound output.

3.8 Reading and Writing Files

This example shows how to write text into a file and retrieve the text afterwards from the file again. If the file does not exist, it is newly created.

```
1 path = "/sdcard/"
2 name = "foo.txt"
3 CALL CHAROUT path || name, "Written to file at" DATE() TIME()
4 CALL CHAROUT path || name, "Some interesting text"
5
6 text = CHARIN(path || name,1,1000)
7 SAY text
```

Listing 8: Write to and read from file

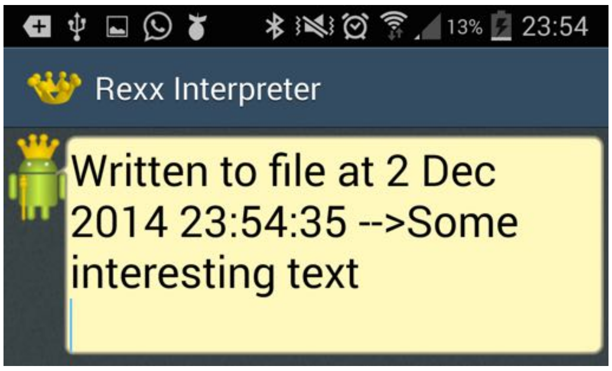
In the first two lines of the script, the path and the name of the file are defined. After that, in line 3 the file is created and some text, inluding date and time, are written into it. As can be seen, the first argument specifies the file path and name, while the second argument specifies the string to be written into the file. Line 4 adds more text into the existing file. Note that the file is not deleted and recreated, instead every further call of the charout method will append the specified string to it.

Line 6 retrieves the content from the previously created file and stores the string into the `text` variable. Again, the first parameter specifies the path and filename to be read.

The second parameter defines the index of the first character to be read out. The last parameter tells the script how many characters should be read. It is not mandatory to define the last two parameters, however if they are not supplied, only the first character of the file will be retrieved. In this example, reading starts with the first character of the file and includes up to 1000 characters (which is, in this case, the entire file). Line 7 finally prints the file's content.

Note that further executions of the script, without in between deletion of the created file, will append the specified strings, i.e. the file will not be deleted automatically and the printed text will include the text created in previous executions.

Figure 14 shows the outcome of the script. The text is first saved to a file "foo.txt" and, after that, readout and displayed.



Written to file at 2 Dec 2014 23:54:35 -->Some interesting text

Figure 14: Text read from file

3.9 Sending Keyevents

The following example shows how to use keyevents.

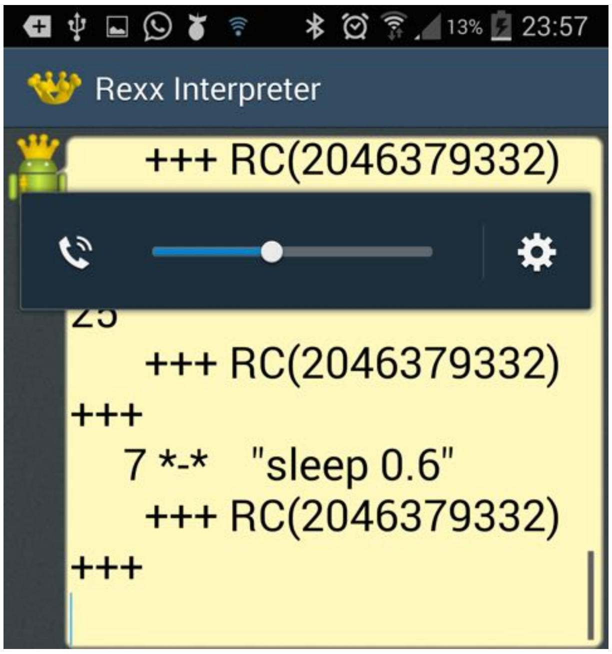
```
1  DO 5
2      "input keyevent 24"
3      "sleep 0.6"
4  END
5  DO 5
6      "input keyevent 25"
7      "sleep 0.6"
8  END
```

Listing 9: Send keyevent

It can be seen that keyevents can easily be triggered by using "input keyevent <number>" as in line 2 and 6. The keyevents used in this example are 24 (increase volume) and 25 (decrease volume). The events are fired 5 times each, having a delay of 0.6 seconds in between. The "sleep <seconds>" call uses the android sleep function, instead of Rexx's. This is due to the fact that the Rexx built in sleep function results in an error. Android's sleep function works well.

Keyevents can be almost anything, ranging from volume buttons, the home button, letters and numbers to the camera or the caps lock key. For a list of all keyevents visit `http://developer.android.com/reference/android/view/KeyEvent.html` [see Andr14c].

Figure 15 shows the script in action. The loops create and execute the input and the sleep function calls. In the upper half of the figure, a change in the volume can be seen (which is caused by the calls of the keyevent function).



```
      +++ RC(2046379332)
25
      +++ RC(2046379332)
+++
    7 *-*   "sleep 0.6"
      +++ RC(2046379332)
+++
```

Figure 15: Keyevent volume

3.10 Opening the Browser

This short example shows how to open an URL in a browser.

```
SAY "Which url would you like to visit?"
PARSE PULL url
"am start -a android.intent.action.VIEW --user 0 -d http:" || url
```

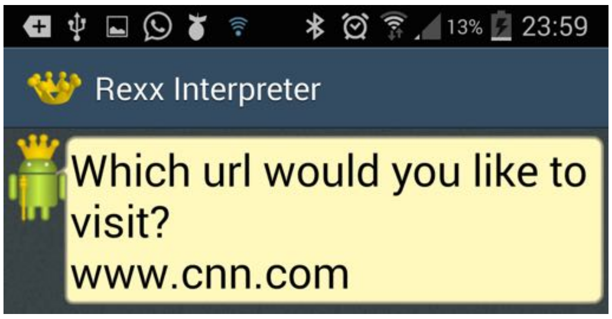
Listing 10: Open the browser

In line 1 and 2 the user is asked for the url to be visited. It is then stored in the `url` variable.

Line 3 starts the activity with the following settings:

- **-a android.intent.action.VIEW**: Defines the action to be called.
- **--user 0**: Specifies the user (Without this, there will be an exception).
- **-d http: || url**: Specifies the intent's data URI. The URI follows the following pattern: "http:<url>", where <url> is to be replaced with e.g. "www.wu.ac.at".

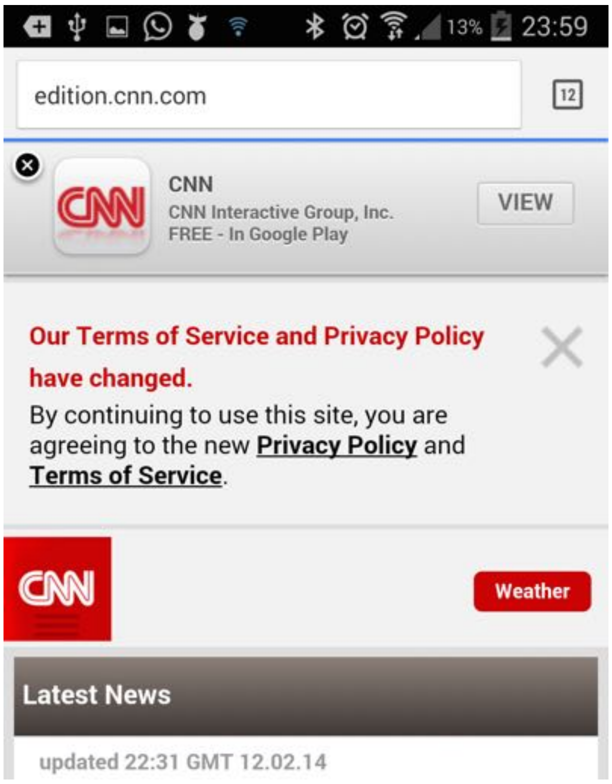
Figure 16 shows the script in action. As can be seen, the user chooses www.cnn.com to be displayed.



Which url would you like to visit?
www.cnn.com

Figure 16: User chooses url

After the desired url is entered, the browser opens and the specified website is displayed (see Figure 17).



Our Terms of Service and Privacy Policy have changed.
By continuing to use this site, you are agreeing to the new Privacy Policy and Terms of Service.

updated 22:31 GMT 12.02.14

Figure 17: www.cnn.com

4 Limitations of Rexxoid

At the time of writing (fall 2014) there seems to be no documented Rexxoid script, which starts an activity and, after that, executes any further command. When trying to do so, the script crashes with the infamous message "Unfortunately, Rexx has stopped". To illustrate the issue the following example is provided.

```
"am start -a android.intent.action.VIEW --user 0 -d 'geo:0,0?q=vienna'"
"sleep 10"
"am start -a android.intent.action.VIEW --user 0 -d 'geo:0,0?z=16'"
```

Listing 11: Open Google Maps and zoom in

This should open Google Maps showing Vienna. Then, after a 10 seconds break, the zoom level should be changed to 16.

While the first two lines perform as expected, the script crashes at the third one. However, when sending these commands via the Android Debug Bridge (adb), they work well. This issue is also posted in the Google group comp.lang.rexx. By fall 2014, there is no solution to this.

Another limitation of Rexxoid is that text or any other result produced by the Android system cannot be retrieved and stored into a Rexx variable. Instead, it can only be displayed as output. This heavily reduces the number of use cases of Rexxoid.

Unfortunately, the Rexxoid community seems to be limited to a very small group of users. Probably, this is the result of other implementations (e.g. BRexx) of Rexx running on Android, which provide more features than simply executing Android shell commands. The small Rexxoid community also results in very few available Rexxoid script examples. Rexxoid comes with some examples preinstalled, but only one of them actually includes a call to the Android shell.

Note that the author did not reach Rexxoid's author to clarify if these limitations are actually results of errors or incompletions of Rexxoid or if there are simply no documented examples for these issues.

5 Rexxoid versus BRexx

Rexxoid and BRexx are both interpreters of Rexx for Android. While BRexx builds on the scripting layer for Android (SL4A), Rexxoid does not make use of it. As only BRexx uses this functionality, SL4A is considered to be a part of BRexx for the comparison. In this section they will be compared and major differences highlighted.

Both applications were tested on the same device, Samsung Galaxy S4 (model number GT-I9505) running Android version 4.4.2. Therefore there should be no differences owing to the used device. All statements refer to the time of writing, which is fall 2014.

This section was written in cooperation with Eva Gerger [see Gerg14].

5.1 Comparison

Installation The Rexxoid application can be downloaded directly from the Google Play store. Android devices install the downloaded application automatically and after that the installation process is finished. BRexx needs two applications, SL4A and BRexx itself, neither of them can be downloaded from the Google Play store. They have to be obtained from external sources. This requires also the permission to install applications from third parties, which is not necessary for Rexxoid.

Example repository Rexxoid comes with a few preinstalled examples. Of these few examples only one uses Android functionality. Also there are no further examples for Rexxoid on the web which use Android functionality. In comparison, BRexx also comes with only a few preinstalled examples. However, some more can be found on the web. Admittedly, those are example scripts in other languages (e.g. in Python or JavaScript), but as they also use SL4A functions they can be adapted quite easily.

Functionality The functionality of Rexxoid is very limited. Besides "normal" Rexx code, only Android shell commands can be used. Hardly any predefined functions are available. BRexx, on the other hand, offers a great variety of functions. Almost any Android component can be addressed in BRexx (e.g. sensors can be utilised). Additionally, in contrast to Rexxoid, BRexx scripts can run in the background, which leads to even further fields of application.

Although both applications come with a small set of granted Android permissions, for some functions additional permission are necessary. Adding permissions is rather effortful in both cases, it can only be done by editing the source code of the application (specifically, the Manifest file) and therefore requires a reinstallation. BRexx has the advantage that written scripts are not deleted in this process, while Rexxoid's need to be saved in advance.

Usability Rexxoid's user interface is very slim and the navigation is rather inconvenient (e.g. the script has to be opened in editing mode before it can be executed). BRexx's interface is more sophisticated and offers handy options (e.g. Save & Run). Also the menus are very well structured and useful. This makes getting started very easy.

Performance Both applications come with the same script to measure their performance (Average REXX clauses-per-seconds by Mike Cowlishaw). When executing the script, Rexxoid ends up with about 450.000 clauses per second, whereas BRexx reaches about 1.1 million. It can be seen that BRexx is about twice as fast as Rexxoid.

Community Rexxoid's community consists only of a handful of people. BRexx's community is actually also very little, but because of the big SL4A community support for many of the upcoming problems can be found in this community as well.

Documentation There is no documentation of Rexxoid. BRexx does not offer any documentation as well. But at least SL4A offers a list of available functions and their arguments. Unfortunately, there is no information on dependencies of the functions (i.e. function x has to be called before function y or y will not work). Nevertheless, the built in API browser is comfortable to use.

Readability Reading and understanding Rexxoid scripts is quite difficult. Android shell commands are very long and not self-explanatory. Due to very meaningful function naming in SL4A, BRexx scripts are easy to read and understand.

Debugging Debugging in Rexxoid is easier than in BRexx. Exceptions thrown by the operating system or by Rexxoid itself are displayed in Rexxoid's default output console. Syntactical errors are noticed immediately and prevent execution of the script. BRexx does not display any errors by default. After execution of a script, the unfiltered logcat output can be displayed (which is really long and not overseeable). So the best way of displaying errors and debugging is to connect the device to a computer and use the Android Debug Bridge to display a filtered logcat output.

Software updates The newest version of Rexxoid is from September 2014, while BRexx's newest version is from March 2013. Rexxoid is being developed continiously, BRexx has longer release cycles.

5.2 Recommendation

Table 2 summarises the results of section 5.1. The application which performs better in a specific aspect is marked with a plus sign (+), the less well performing application marked with a minus sign (–). If they are equal, a tilde is used (~).

Aspect	**Rexxoid**	**BRexx**
Installation	+	−
Example repository	−	+
Functionality	−	+
Usability	−	+
Performance	−	+
Community	−	+
Documentation	−	+
Readability	−	+
Debugging	+	−
Software updates	~	~

Table 2: Comparison of Rexxoid and BRexx

As Table 2 indicates, BRexx is the better choice in almost any perspective. Maybe Rexxoid improves and enriches its feature set in the future, but currently the clear recommendation is BRexx.

6 Conclusion

Rexxoid enables developers to write short Rexx scripts for Android devices easily. However, if Android functionality should be used as well, it becomes complex and the functionality is very limited. Unfortunately, Rexxoid can not run in the background, which makes it unusable for many tasks.

The Rexxoid community is really small, which makes it even harder to find support. There are almost no nutshell examples available on the web. Additionally, hardly any scripts are preinstalled so it is tricky to get started with Rexxoid.

What makes it even more difficult to write scripts is the fact that there is not a single line of documentation available. Of course, there is a documentation for Rexx itself, but not for the interaction with Android.

The other approach to run Rexx on Android discussed in this paper, BRexx, seems more practical, comes with a greater community and a documentation. Furthermore, BRexx includes features that are absolutely needed in Rexxoid, especially scripts running in the background.

Maybe the nutshell examples in this paper help someone to fix a problem or develop new scripts. At the current stage, one should definitely consider using an alternative.

References

[Andr14a] Android.: Android Debug Bridge. http://developer.android.com/tools/help/adb.html, Accessed on 2014-12-23.

[Andr14b] Android.: Installing the Android SDK. http://developer.android.com/sdk/installing/index.html, Accessed on 2014-12-23.

[Andr14c] Android.: KeyEvent. http://developer.android.com/reference/android/view/KeyEvent.html, Accessed on 2014-12-23.

[Butl11] Butler, Margaret: Android: Changing the Mobile Landscape. In: IEEE Pervasive Computing 10 (2011) 1, p. 4-7.

[Flat13] Flatscher, Rony G.: Introduction to REXX and ooRexx. Facultas, 2013.

[Gerg14] Gerger, Eva.: BRexx: Running Rexx on Android Systems. Vienna University of Economics and Business, Seminar Thesis, 2014.

[Goog14] Google Play.: Rexx for Android. https://play.google.com/store/apps/details?id=com.jaxo.android.rexx, Accessed on 2014-12-23.

[Inte14] International Data Corporation.: Smartphone OS Market Share, Q3 2014. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, Accessed on 2014-12-06.

[Open14] Open Handset Alliance.: Industry Leaders Announce Open Platform for Mobile Devices. http://www.openhandsetalliance.com/press_110507.html, Accessed on 2014-12-06.

[Wiki14] Wikipedia.: Intent (Android). http://en.wikipedia.org/w/index.php?title=Intent_(Android)&oldid=626240352, Accessed on 2014-12-23.