Wirtschaftsuniversität Wien

Abteilung für Wirtschaftsinformatik

LV-Nr.: 0974 WS 2006

Vertiefungskurs VI: Projektseminar

LV-Leiter: Univ. Prof. Dr. Rony G. Flatscher

Term Paper

Automated email list analysis with Open Office.org

Author:

Istvan Szilagyi, Matr. Nr. 0250796

Table of Contents

Illustration Index

1 Abstract

Object of this paper is the finding and description of a solution for automation tasks for the city of Vienna.

The main scope is on finding an interoperable (Windows, Linux) and easy to use solution to provide time saving automation for Open Office, which also is easy to maintain and to deploy.

1.1 Task Description

Underlying task for this work is a workflow, which has to be completed manually. The department for information and communication techniques of the city of Vienna is the host of all the email accounts used in all the municipalities of the city.

Therefore more thousand accounts have to be analyzed month for month regarding their size and the usage data. Users, who exceed their mail account limit, must be warned via an email.

This basic task is complicated with a bunch of rules, which must be applied at the department, and is a task clearly determined for automation.

The description of the workflow is the following:

1.) A list of all users and their account information, including the maximal size of their account and the actual usage, is generated by the mail server, each month.

2.) This table has to be analyzed and split into four different groups, each saved differently, namely:

- Usage exceeds the capacity of 80 MB, must be shown with grey background
- Usage exceeds a capacity under 80 MB, must be shown with red background
- Usage exceeds a capacity over 80 MB, must be shown with blue background
- List of accounts with a limit higher than 80 MB, which must be shown in a separated list (even without exceeding the limit)

3.) For every department an own list must be generated, the lists of the departments is to be found in a help list. A second help lists shows the relation between the accounts and the departments.

4.) After this, the head of the department is to be found in the second help list and an email has to be generated and sent with the table of exceeders attached.

5.) For documentation purposes a copy of this email has to be sent to the View4 server (every order has its own email address)

Considering the planned switch from Microsoft's proprietary products to open source solutions, the department of information and communication techniques wanted to use a solution which is able to run on both Windows and Linux systems, automating OpenOffice.org which is being installed on all the machines used by the city of Vienna.

A solution with OORexx has been chosen, which is going to be described in this paper.

2 System Description

This section should give a brief overview about the technologies used for the technical solution of the problem described in the abstract above.

2.1 Rexx

Rexx is an abbreviation standing for “Restructured Extended Executor”, and was designed as a “human centric” interpreter for IBM. Mike F. Cowlishaw first developed it in the year 1979. Since then a lot of development and addition have been made to the interpreter, but the basic syntax stayed the same, which should be visualized with some basic snippets. [FLAT01]

2.1.1 Basic Syntax

Being a human centric interpreter, the language structure is kept simple and Rexx does not need any kind of variable declaration. It handles only string data types and is case-insensitive. If it confronts with a command not known by the interpreter it simply handles it over to the program, which executes the interpreter.

In our first snippet, we simply print the well-known “Hello world!” on the display and remove all files recursively in the order.

```
say "Hello world!"
"rm -rf *"
```

Figure 1 Rexx Hello World

Rexx neither features an array kind of data type, but the official recommendation is to use variable names extended with numbers, which than look and behave like an array. This is shown in figure 2, where an “array” is created with to values which are then given back on the command line.

```
array.1 = “First Item”
array.2 = “Second Item”
l = 2

do i=1 to l
     say array.i
end
```

Figure 2 Rexx Array

The readability of Rexx is clearly to see.

2.1.2 OORexx

The original Rexx language is a procedural language with no support for object orientation. Therefore in 2004 a non-profit oriented group of scientists took over the Rexx source code, and is since then responsible for the deployment of OORexx, the object oriented enhanced version of Rexx. Since then, under the wings of the Rexx Language Association (RexxLA), Open Object-oriented Rexx has been made available for almost all platforms.

The language structure stayed the same, but some new functions have been introduced. The “twiddle” is used as message operator, classes and methods are shown with two colons.

OORexx features all common object oriented aspects, and also features an easy to implement multiple inheritance.

```
.Liberalist  ~new("Roosevelt") ~speak /*create a liberalist, let
him speak*/
.Communist ~new("Lenin")~speak   /*create a communist, let him
speak*/
::class  Liberalist
::method init  /* constructor method */
  expose name  /* establish direct access to attribute (object
variable)*/
  use arg name /* retrieve argument, assign it to attribute  */
::method name attribute /* define set and get attribute methods
*/
::method speak
  say self~name":" "I believe in capitalism."
::class  Communist subclass Liberalist
::method speak
  say self~name":" "I believe in it too, but I pretend not to!"
```

Figure 3 OORexx example

This snippet shows easy to follow examples how the message operator and the colons are working. It shows the following output on the screen: „Roosevelt: I believe in capitalism. Lenin: I believe in it too, but I pretend not to!" The method of speaking is being overwritten in the subclass.

Figure 4 [FLAT01] shows how inheritance can be made. Here we mix the classes of a road and a water vehicle to get an amphibian vehicle. It is also shown that the interpreter can automatically create the "getter" and the "setter" methods for an attribute within a class.

```
/* Multiple Inheritance */
.RoadVehicle     ~new("Truck")  ~drive
.WaterVehicle    ~new("Boat")   ~swim
.AmphibianVehicle~new("SwimCar")~show_off
::CLASS  Vehicle          /* define the vehicle base class */
::METHOD name ATTRIBUTE /* let interpreter define a getter and
setter method  */
::METHOD init             /* define constructor method  */
  self~name=ARG(1)        /* use the setter method to set the at-
tribute's value */
::CLASS  RoadVehicle     MIXINCLASS Vehicle
```

```
::METHOD drive          /* define a road vehicle method */
  SAY self~name": 'I drive now...'" /* use the attribute getter
method  */
::CLASS  WaterVehicle    MIXINCLASS Vehicle
::METHOD swim           /* define a water vehicle method  */
  SAY self~name": 'I swim now...'"  /* use the attribute getter
method    */
::CLASS AmphibianVehicle SUBCLASS RoadVehicle INHERIT WaterVehi-
cle
::METHOD show_off       /* demonstrate multiple (implementation)
inheritance  */
  self ~~drive ~~swim   /* using cascading messages (two twid-
dles) */
/* yields the following output:
  Truck: 'I drive now...'
  Boat: 'I swim now...'
  SwimCar: 'I drive now...'
  SwimCar: 'I swim now...'
*/
```

Figure 4 Inheritance [FLAT01]

2.2 OOo

OOo is an abbreviation, standing for Open Office org, an open source office application project available on the website, which is hinted on in the name of the product.

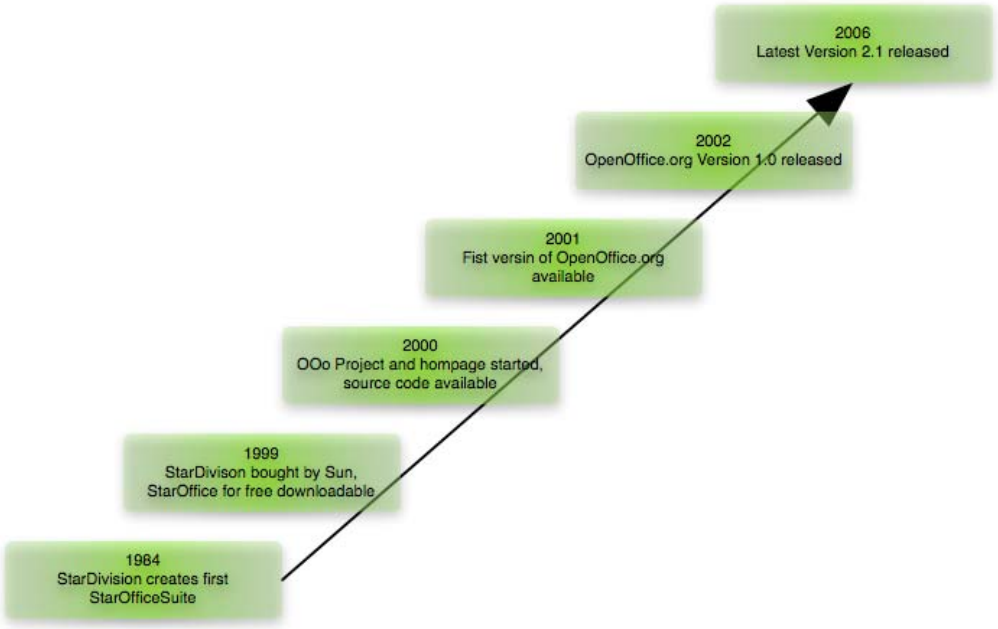

Figure 5 Ooo Evolution

Figure 5 illustrates the historical milestones of Open Office.org, being at its latest version 2.1 in 2006. The first Version was available for download 2001 after the source code for Sun Microsystems Staroffice, originally developed by a German company since the mid 80-ies, was opened, and the OpenOffice.org homepage was created, with the project for the creation of an open source office program. Since then Sun Microsystems and other companies deliver the biggest parts to the development for Open Office. [SUN01]

Open Office.org is distributed under the LGPL (GNU Lesser General Public License). This allows developers to sell their developments made with the source code, available for public; they are also allowed to use the binaries for commercial usage.

Of course this does not include the distribution of the SUN Star Office, as it is a product of SUN Microsystems, sold under a different licensing contract.

When given the source code for public, Sun removed all proprietary components of the product, and also components of third parties which can not be used in an open source development. That’s why Star office has some features,

mostly containing fonts, templates and databases, which Open Office does not feature. [SUN02]

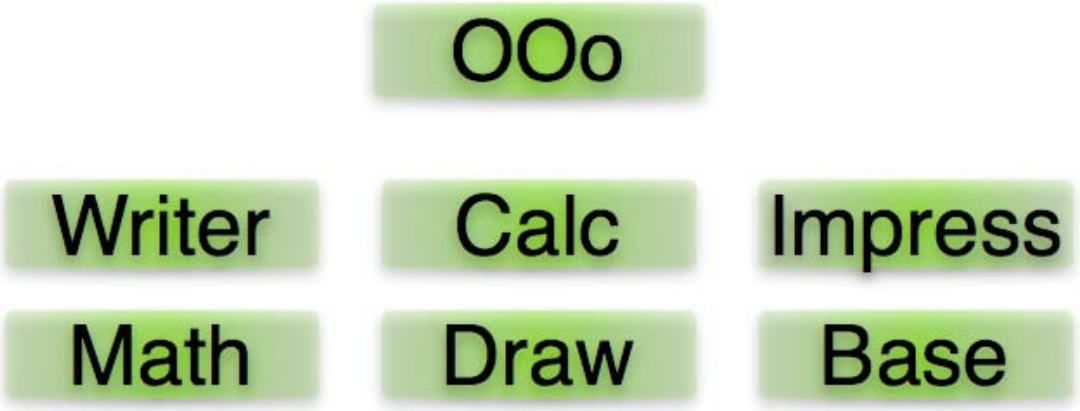

Figure 6 OOo main parts

Figure 6 shows the main parts of Open Office.org. All components of Open Office use the vendor independent file formats of OASIS. [OASIS1]

The Writer is the text-processing program in the OOo Package. It has powerful tools and wizards for the creation of a large variety of documents. It is fully compatible with the formats of Microsoft's Word, and also provides support for PDF and HTML.

Calc is the spreadsheet-handling unit within OOo. It also features full compatibility with Microsoft's competing product Excel as it can read and write Excel files. You can find easy to use wizards, support for natural language forms and scenario managers. [CALC01]

Impress lets you create multimedia presentations. Under the exporting possibilities, besides the Open Document format one will also find the possibility for Microsoft PowerPoint, and also Flash movies can be created from a presentation.

For writing scientific and professional equitation Math can be used. It makes possible to draw such equitation easily and place them right into other Open Office components such as the text-processing program Writer.

Draw supports working with graphics. It allows you to draw diagrams and any kind of graphics. A clip art gallery is included, as the possibilities to read and export in almost every kind of known graphical format. Here again a flash exporting possibility is provided.

Base is the database program for data manipulation. For basic usage it comes with a java based, XML-storing database engine, called HSQL. It allows you to manipulate data with SQL und simple wizards, or design views. Also for more advanced requirements it features connection possibilities for Access, MySQL and other popular databases, and is also compatible with any ODBC or JDBC database drivers.

2.3 BSF4Rexx, UNO

The key for being able to automate Open Office simply with a Rexx are the underlying techniques to connect Rexx to Java and Java to Open Office.

2.3.1 BSF4Rexx

BSF stands for Bean Scripting Framework.

“Bean Scripting Framework (BSF) is a set of Java classes which provides scripting language support within Java applications, and access to Java objects and methods from scripting languages.” [BSF01]

This means, the BSF allows scripting programs to access java classes and all their methods after importing. There are several different BSF versions and programs for the different kind of popular scripting languages like Jacl for Tcl, the version used to let Rexx access any kind of java classes and their methods is BSF4Rexx.

The Evolution until the current version “The Vienna Version” can be seen in figure 7.

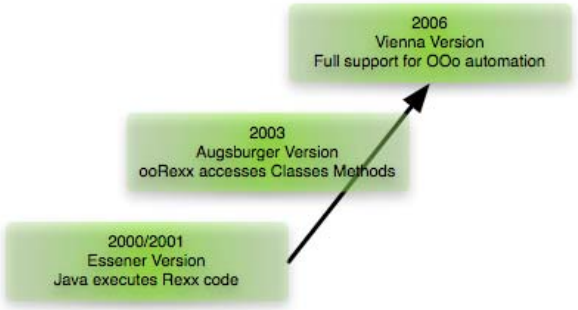

Figure 7 BSF Evolution

Figure 8 provides information about the general architecture of BSF4Rexx. [FLAT02]

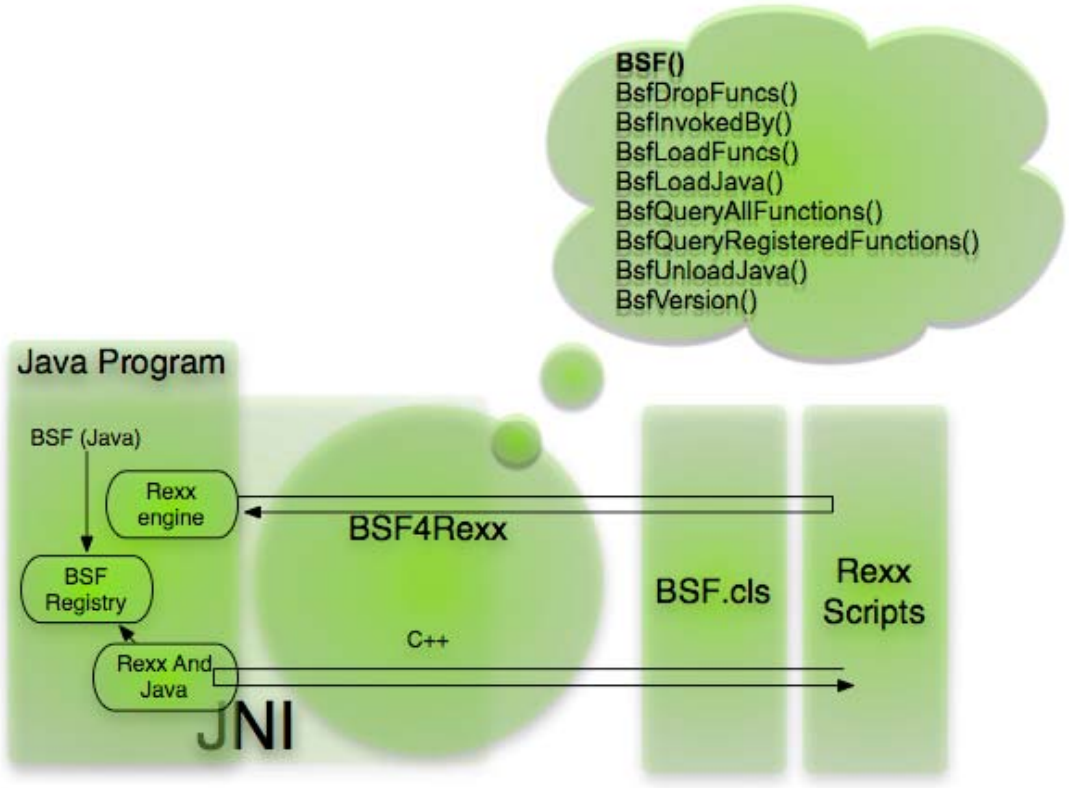

Figure 8 BSF4Rexx communication [FLAT02]

This means, that after importing the BSF.cls file, you have the ability to import any kind of java classes and use them like they are described in the java API.

As the later solution uses a lot of features of BSF, here only a small snippet is provided to demonstrate to way of working with BSF. Figure 9 [FLAT04] shows, how to require the BSF.cls file and how to use it to tell the user the actually installed java version.

```
/* "getJavaVersion.rex": classic Rexx version, querying the in-
stalled Java version */
say "java.version:" bsf('invoke', 'System.class', 'getProperty',
'java.version')
::requires bsf.cls  /* load the Java support */
```

Figure 9 BSF4Rexx example [FLAT02]

2.3.2 UNO

UNO stands for Universal Network Object. It is the underlying component model for OpenOffice.org. UNO components are compiled and bound libraries. These objects must implement certain interfaces, which are described in a language independent interface description language (UNOIDL). The communication takes place with calls between the different components, and is not bound on a location, therefore UNO components can (could) connect to other components via the intranet, internet or just connected machines. [BURGER01]

Furthermore, these components might be implemented and accessed from any kind of programming language, which has an UNO implementation, and an appropriate bridge.

These bridges also allow the communication between different components in different platform surroundings. The communication can take place via sockets (like TCP/IP) or pipes.

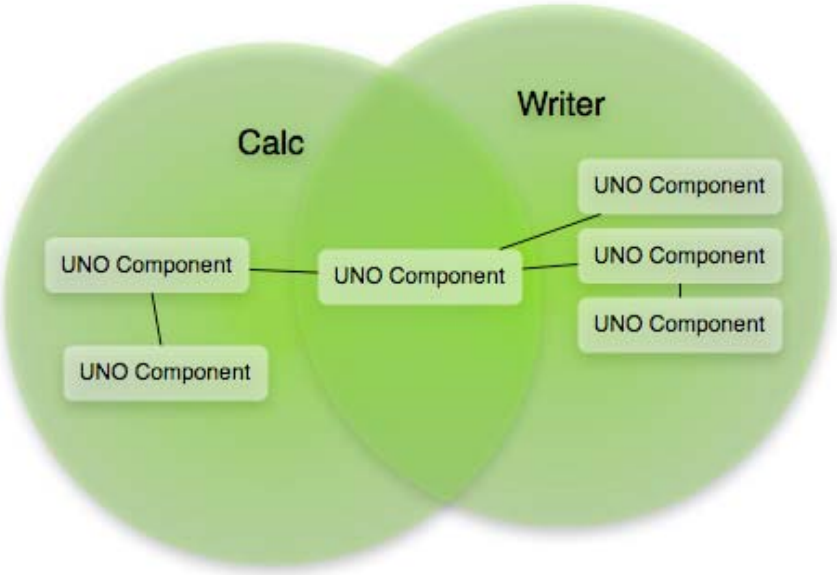

Figure 10 Component usage in UNO

Within Open Office.org there are different components for each application and there are components serving different applications. This means that there is a high reusability between the components. As an example there are specific components for Calc while the printer component is shared between all other applications. This is demonstrated in figure 10.

A service manager is the root object for connections to UNO and serves as an entry point for every UNO application. It is used to instantiate services by their service name, to enumerate all implementations of a certain service and to add or remove factories for a certain service at runtime. The service manager is passed to every UNO component during initialization. The main interface of the service manager is the “com.sun.star.lang.XMultiServiceFactory” interface. It offers three methods: “createInstance()” , “createInstanceWithArguments()” and “getAvailableServiceNames()” [DEVEL01, p. 88]

When a component needs more functionality or information than the central service manager can provide, it can supersede the service manager to get its own component context service managers. [DEVEL01, p. 88f]

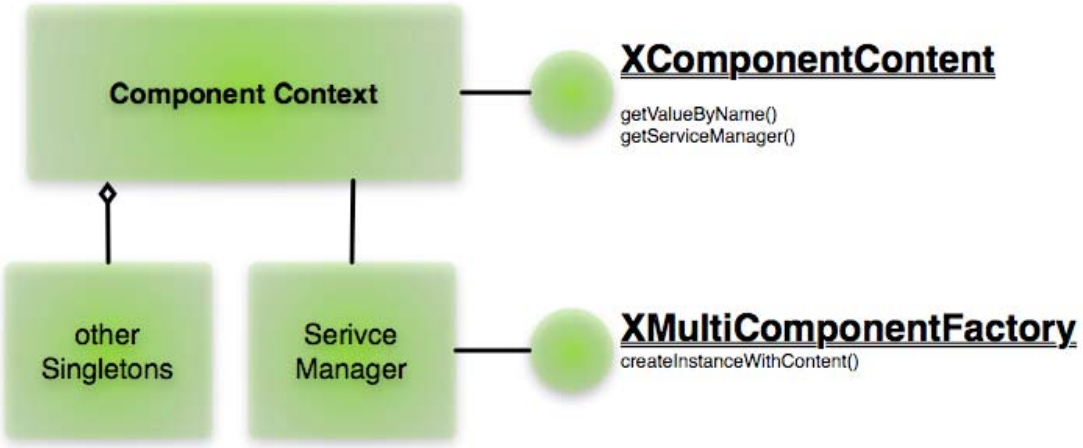

Figure 11 Component Context

3 Solution Description

The resulting script for this work is covering many different parts of BSF4Rexx. It includes not just the Open Office automation, but also features a small java swing based user interface, and also refers to the local, standard email sending program.

With BSF4Rexx it was possible to find a compact and easy to maintain solution, as the program is clearly structured and the syntax is almost completely human readable.

It might be of interest, that in order to achieve the skills for writing the automation scripts for a task with this complexity, it only took 4 weeks to learn the all-over syntax of Rexx and to be able to deal with Open Office.

With the help of small snippets, the general layout of the program and the clarity of Rexx should be visualized.

3.1 Global variables

As a start for our automation, some kind of global variables have to be set, and also the folder for the created documents must be created.

```
datum = date("S") /*todays datum*/
FolderName = "C:\Analyse"datum /*working folder name*/
```

Figure 12 Snippet A 1 - variables

This shows how easy it is to declare variables which then can be reused all over the script. Here the name of the folder we want to work in has been created with today's date in the form "YYYYMMDD".

As Rexx is able to pass on unknown commands to the program, which executes it, the following line is simply passed to the commander and results in creating the desired folders.

```
mkdir FolderName /*creating the folder*/
```

Figure 13 Snippet A 2 – Folder creation

In addition, the packages for Java and UNO support have to be loaded.

```
::requires BSF.CLS --Java support
::requires RXREGEXP.CLS –support for regular expressions
::requires UNO.cls --UNO support
```

Figure 14 Snippet A 3 – BSF and UNO support

These commands are at the end of the script.

3.2 User interface

As mentioned above, java swing was used to create an interface to inform the user about the status of the program, and to provide possibilities for human interaction with the program, when desired.

This was necessary as the program might take a long time with ordering the rather big lists, and also there are numerous possibilities for failures, which should not cause the program to crash, as they are mostly easy to correct, like bad abbreviations in the linking tables.

To be able to use a java class in Rexx, the java class has to be imported at the beginning of the program.

```
.bsf~bsf.import("javax.swing.JLabel","JLabel")  /* import the
Java Swing JLabel class */
```

Figure 15 Snippet B 1 – swing import

Snippet B1 illustrates, how the JLabel class can be imported into BSF. After this line of command, this java class is available for usage with all its methods and interfaces. A name is always assigned to it, in this case “JLabel”, under which Rexx can identify the class.

After importing all desired classes of swing the same way we have just illustrated, we are able to create our information panel. We can easily do so by defining the objects, and then placing them into the desired layout.

```
label= .JLabel~new("");

frame = .JWindow~new();

icon = .Icon~new("ma.gif");

frame~getContentPane()~setLayout(.GridBagLayout~new());

frame~getContentPane()~~add(labelp)~~add(action);
```

Figure 16 Snippet B 2 – setting up the message box

It was in the interest of the user to create an information panel, which cannot be closed, and also always stays on top of the screen, as there is no need for human interaction during the program is running, except of errors.

Therefore a position for the window must be calculated, which, in a Windows environment, is the best right above the task bar, in the right corner.

For this, the overall dimensions of the display have to be emitted, and the right position has to be calculated.

Such a function is provided by java; it simply calculates the dimensions of the screen minus the place the task bar needs.

```
ge = .GraphicsEnvironment~getLocalGraphicsEnvironment()
maximumWindowBounds = ge~getMaximumWindowBounds();
h  = maximumWindowBounds~toString()
```

Figure 17 Snippet B 3 – getting the actual screen size

This information then is parsed into the variables „resx“ and „resy“, standing for resolution x and resolution y. After this we are ready to calculate the right position and just need to show our window. (Screenshot 1)

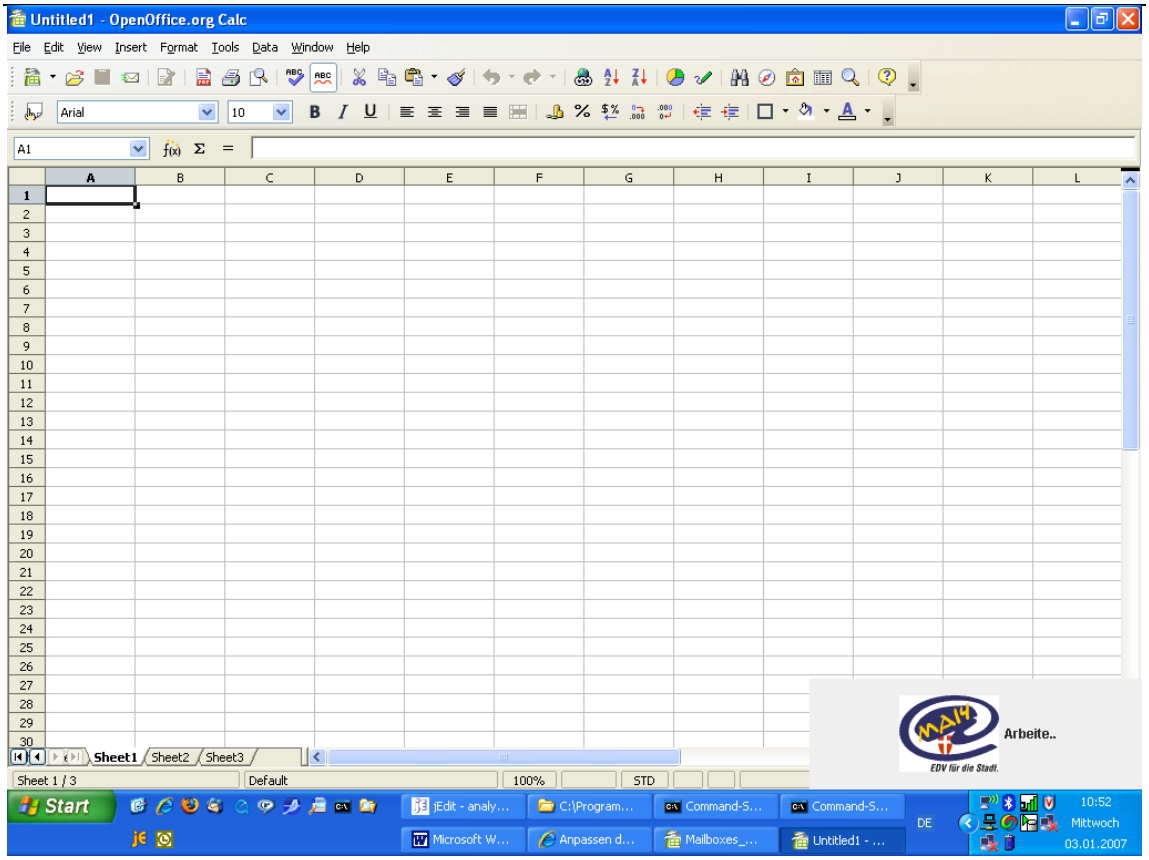

Figure 18 Screenshot 1 – Screen with message box

3.3 Open Office

For the main automation task, open office must be loaded and several documents have to be loaded, as the splitting of the tables is done.

To load an Open Office document, and also to create new ones, the desktop object must be emitted to connect to UNO.

```
xcomponentLoader = UNO.createDesktop()~XDesktop~XComponentLoader
```

Figure 19 Snippet C 1 – connecting to the desktop

For the first workflow step, namely the analysis of all email account regarding their size and exceedance, we need two tables, the one containing the information, and a new one for sorting out the table.

```
xCalcComponent = xcomponentLoader~loadComponentFromURL(url,
"_blank", 0, .UNO~noProps)
/*Open a new File for the results*/
newCalcComponent = xcomponentLoader~loadComponentFromURL
("private:factory/scalc","_blank", 0, .UNO~noProps)
```

Figure 20 Snippet C 2 – creating new calc documents and opening

After this we are getting the first sheets of the tables, and we create a new one in the result table.

```
xSheet = XDocument~getSheets~XIndexAccess~
getByIndex(0)~XSpreadSheet
```

Figure 21 Snippet C 3 – getting the right sheet

```
fullList = xSheets~insertNewByName("User mit Limit überschrei-
tung",0)
```

Figure 22 Snippet C 4 – inserting new sheet

With a direct call on UNO we set the cell values. First we just set the headers in the result table, but this function is also used later for filling in the variables into the cells.

```
CALL UNO.setCell newSheet, 0,0, "Rechner"
```

Figure 23 Snippet C 5 – inserting data with „UNO.setCell“

For the first sorting the abort criteria is if the program confronts with an empty line. After an empty line there is no search for any other data in the table.

For comparing, the values are read in from each line from the according cells with the command shown in snippet C 6.

```
Rechner = XSheet~getCellByPosition(0,x)~getFormula()
```

Figure 24 Snippet C 6 – reading out data

Also considering the next sorting step an array with all the departments is created. As there are no real arrays in Rexx, a variable is used with a counter extended. If a new department is found, the counter is simply raised by one and added to the variable name.

```
if DSTList.k <> DST then do
                  k = k+1
                  DSTList.k = DST
                  action~setText("Neue Diensstelle: "||DST)
            end
```

Figure 25 Snippet C 7 – extending the array with items

In snippet C 7 the last item of the Department (Dienststelle, DST) is compared to the Department name just found in the table. If the department does not exist, it is added to the list, and the user is informed in the information panel about the new department found.

Depending on which exceedance was found, the data must be filled in, like described in the workflow. For this, the background property of each cell in the whole line must be set to the according color with the command shown in snippet C 8.

```
      if Limit = 80 then do
            NewRechnerCell~xPropertySet~setPropertyValue
("CellBackColor", box("int","CCCCCC"x~c2d))
```

Figure 26 Snippet C 8 – setting background values

Now the line is ready to get filled with the data just read out, and this is done with the UNO command setCell like in snippet C 5.

After inserting the colored data, the account information is selected which has to be shown on a new spreadsheet, containing only the accounts with a limit higher than 80 MB.

If the aborting criterion is reached, the open spreadsheet has to be closed and saved into the working directory, and the tables are closed.

```
      if Rechner = "" then do
            i = 2
            /*document saving in working path and ending sort*/
            StoreLimit = newDocument~XStorable
            Path = FolderName"\Limit_Mailboxes".ods
            SaveUrl = ConvertToURL(Path)
            StoreLimit~storeAsURL(SaveURL, .UNO~noProps)
      end

/*closing documents*/
newCalcComponent~dispose()
xCalcComponent~dispose()
```

Figure 27 Snippet C 9 – saving and closing

The two next workflow steps are solved in one loop, which causes a complex structure.

In the first structure, the numbers used internally by the department for information and communication techniques have to be found, which relate to the “Dienststellen” saved in the array, as described before. This internal number will then provide the connection to the custodian of the “Dienststelle” and his email address, where the according table has to be sent to.

For this, the table with the classifications between internal number and department name is opened, and searched for the term, which is saved in the array. During the development of this program, it has been realized that the department for information and communication sometimes uses wrong abbreviations for the other departments. This would have caused the program to search endlessly, or to fail. Therefore it was agreed on an end line at the end of the table, which indicates the program to stop and ask for the right abbreviation.

```
do while Reln < 1
      RelDST = RelationsDocument-
Sheet~getCellByPosition(1,Relx)~getFormula()
      if RelDST = SearchDST then do
            PSP = RelationsDocument-
Sheet~getCellByPosition(0,Relx)~getFormula()
            Reln = 3
      end
      /* "ENDE" is aborting criteria. User intervention necessary
to find according data*/
      if RelDST = "ENDE" then do
```

```
            action~setText("Hinweis beachten!!")
            input = .bsf.dialog~inputBox(SearchDST || " nicht ge-
funden. Richtigen Kürzel eingeben:", AnaDST)
            SearchDST = input
            Relx = 0
      end
      Relx = Relx + 1
end
```

Figure 28 Snippet C 10 – loop with aborting criteria

In the next step the custodians with their email addresses have to be found. This happens with almost the same looping technique as shown in the snippet before.

After we have found the custodian and his email address, a new table for the department is created.

```
      /*new file for each departement*/
      DSTComponent = xcompo-
nentLoader~loadComponentFromURL("private:factory/scalc","_blank"
, 0, .UNO~noProps)
      DSTDocument = DSTComponent~XSpreadSheetDocument
      DSTSheets = DSTDocument~getSheets()
      DSTList = DSTDocu-
ment~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet
```

Figure 29 Snippet C 11 – creating new files for the departments

In the next loop, the result table of the first loop is opened and the records are analyzed for their department, and sorted into the right table.

```
      /*filling table with data*/
      do while DSTi < 1
            Rechnerk = SortedDocument-
Sheet~getCellByPosition(0,DSTx)~getFormula()
            Kurz = SortedDocument-
Sheet~getCellByPosition(1,DSTx)~getFormula()
            Name = SortedDocument-
Sheet~getCellByPosition(2,DSTx)~getFormula()
            Plattenplatz = SortedDocument-
Sheet~getCellByPosition(3,DSTx)~getFormula()
            Limitk = SortedDocument-
Sheet~getCellByPosition(4,DSTx)~getFormula()
            DST = SortedDocument-
Sheet~getCellByPosition(6,DSTx)~getFormula()
            if Plattenplatzk = "" then Plattenplatzk = 0
            if Limit = "" then Limit = 0
            if DST = "" then DST = 0
            if DST = AnaDST then do
                        DSTn = DSTn + 1
                        CALL UNO.setCell DSTList, 0,DSTn, Rechnerk
                        CALL UNO.setCell DSTList, 1,DSTn, Kurz
                        CALL UNO.setCell DSTList, 2,DSTn, Name
                        CALL UNO.setCell DSTList, 3,DSTn, Platten-
platz
                        CALL UNO.setCell DSTList, 4,DSTn, Limitk
                        Ueber = Plattenplatz - Limit
                        CALL UNO.setCell DSTList, 5,DSTn, Ueber
                        CALL UNO.setCell DSTList, 6,DSTn, DST
            end
            DSTx = DSTx + 1
            if Rechnerk = "" then do
                  DSTi = 2
            end
```

Figure 30 Snippet C 12 – sorting data

The same way the documents were saved and close at the former workflow step (snippet C 9) the filled document is closed.

These loops are run so many times, as many items the array, containing the department names, has, causing the creation of a table for each department.

3.4 Mail sending

After the tables have been created they have to be sent to the email address of the custodian, and to the email address of the Wiev4 file system, which is used at the department for storage.

As the script should be able to run on different kind of machines, the „simple mail system“ class is used to get the mail-sending program of the user to send the emails. For this, an instance of this class has to be registered within rexx.

```
/*creating mail and attaching created file*/
     SimpleMailSystem   = localSM~
createInstanceWithCon-
text("com.sun.star.system.SimpleSystemMail", localCC)
     XSimpleMailClientSupplier = SimpleMailSys-
tem~XSimpleMailClientSupplier
     XSimpleMailClient = XSimpleMailClientSup-
plier~querySimpleMailClient
     mail = XSimpleMailClient~createSimpleMailMessage
```

Figure 31 Snippet D 1 – connecting to mail system

After this is registered the manipulation of this instance, saved in the variable „mail“, can be done very easily, which is demonstrated in snippet D 2, where the recipient and the subject are set.

```
mail~setRecipient(Mailadress)
mail~setSubject("Überschreitungsliste." || Mailadress)
```

Figure 32 Snippet D 2 – recipient and subject

The according table must then be attached to the mail, here the java class requires an array as parameter, which is created with the according functions of bsf.

```
      attach = bsf.createArray(.bsf4rexx~string.class, 1)
      attach[1] = SaveUrl
      mail~setAttachement(attach)
```

Figure 33 Snippet D 3 - attachment

In snippet D 3 the variable „SaveUrl“ is the variable used to determine the path for saving the created table in the last loop.

The now created message is ready to send with attachment. For this the message must be „marked“ for the email program, which is done with the function „flag“. The marked message is then sent.

```
flag =
bsf.getConstant("com.sun.star.system.SimpleMailClientFlags",
"NO_USER_INTERFACE")

      /*sending message*/
      XSimpleMailClient~sendSimpleMailMessage(mail, flag)
```

Figure 34 Snippet D 4 – sending message

At the department for information and communication techniques, it is not allowed to send automated emails to prevent abuse. For each email the user must confirm the email.

This is granted, as all the email programs used at the department (Microsoft Outlook and Thunderbird) do not allow rexx to send the email without notifying the user about the procedure.

3.5 Conclusion

The department for information and communication techniques is managing approximately 19000 registered email accounts for the city of Vienna. This large amount of records makes this case an ideal situation for automation where the benefit of automation is extremely high.

Even with a high failure rate within the provided data (missing links, bad email addresses) and the need to confirm every single email sent, the work load can be reduced dramatically for the employees of the department.

It is to assume that a work load for about 4-5 days is to be saved after introducing the automated system, and only one workday is needed for the enrolment of the employees.

This is a huge benefit regarding the overall development and installation time of four months, including the time for the developer to get used and known to the Rexx language structure.

Also the soon planned migration from Windows to Linux will not cause any trouble as Rexx is able to automate Open Office on both platforms.

4 References

BSF01 Bean Scripting Framework, The Apache Software foundation, URL (2006-12-05): http://jakarta.apache.org/bsf/

Burger01 Burger Martin, Open Office.org automation with Oorex, 2005, Wirtschafsuniversität Wien (Vienna University of Businnes Administration), Austira.

CALC01 OpenOffice.org, Product description – Calc, SUN Microsystems, URL (2006-12-4): http://www.openoffice.org/product/calc.html

DEVEL01 OpenOffice.org 2.1 Developers Guide, Sun Microsystems, May 2005

FLAT01 Flatscher Rony G, Ressurecting Rexx, Introducing Object Rexx, May 2006, Wirtschaftsuniversität Wien (Vienna University of Businnes Administration), Austria

FLAT02 Flatscher Rony G., The Vienna Version of BSF4Rexx, 2006, Presentation at the 2006 International Rexx Symposium, USA; URL (2007-01-02):

http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf

OASIS01 Open Document Format for Office Applications, OASIS OPEN, 2006, URL (2006-12-04):

http://www.oasis-open.org/committees/office/charter.php

SUN01 About OpenOffice.org, SUN Microsystems, URL (2006-01-2): http://about.openoffice.org/index.html

SUN02 Licensing FAQ, SUN Microsystems, URL (2006-01-2):

http://www.openoffice.org/FAQs/mostfaqs.html

UNO01 Uno Developement Kit Project, SUN Micorsystem, 2006, URL (2006-10-3): http://udk.openoffice.org/

5 Source Code

```
.bsf~bsf.import("javax.swing.JLabel","JLabel")  /* import the Java Swing JLabel class */
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout") /* import the Java Awt Gridlayout class */
.bsf~bsf.import("javax.swing.ImageIcon","Icon")  /* import the Java Swing Imageicon class */
.bsf~bsf.import("javax.swing.JButton","JButton")  /* import the Java Swing Imageicon class */
.bsf~bsf.import("javax.swing.JPanel","JPanel")  /* import the Java Swing Imageicon class */
.bsf~bsf.import("java.awt.GridBagLayout","GridBagLayout") /*import the Java Awt GridBagLayout class */
.bsf~bsf.import("java.awt.Toolkit","Toolkit") /*import the Java Awt Toolkit class*/
.bsf~bsf.import("java.awt.GraphicsEnvironment",  "GraphicsEnvironment")  /*import  the  Java  Awt  GraphicsEnvironment
class*/
.bsf~bsf.import("javax.swing.JWindow","JWindow") /* import the Java Awt Gridlayout class */
.bsf~bsf.import("javax.swing.JOptionPane","JOptionPane") /*import the JOptionPane*/

datum=date("S") /*todays datum*/
FolderName = "C:\Analyse"datum /*working folder name*/
mkdir FolderName /*creating the folder*/

localCC    = UNO.connect()   /* Connect to Local UNO */
localSM    = localCC~getServiceManager   /* create its ServiceManager  */

/*creating frames, buttons, etc. for JAVA interface*/
label= .JLabel~new("");
frame = .JWindow~new();
icon = .Icon~new("ma.gif");
jbuttonp = .JPanel~new();
labelp = .JPanel~new();
text = .JLabel~new("Arbeite..");
action = .JLabel~new("Arbeite..");

jbuttonp~setLayout(.Flowlayout~new(1))
labelp~~add(label)

/* set the the frame to use the GridBagLayout */
frame~getContentPane()~setLayout(.GridBagLayout~new());
frame~getContentPane()~~add(labelp)~~add(action);

/*dimensions of information frame */
boxw = 300
boxh = 100

/*getting desktop size to position the box*/
ge = .GraphicsEnvironment~getLocalGraphicsEnvironment()
maximumWindowBounds = ge~getMaximumWindowBounds();
h   = maximumWindowBounds~toString()

/*parsing desktop size */
parse var h token.1 '=' token.2 '=' token.3 '=' token.4 '=' token.5 '='
parse var token.4 resx ','
parse var token.5 resy ']'

/*positioning the box*/
boxpx = resx - boxw
boxp = resy - boxh

/* Add eventhandling */
frame~bsf.addEventListener('window', 'windowClosing', 'call BSF "exit"')
frame~setLocation(boxpx,boxp); /* set the location of the frame on the screen */
frame~~pack()~~setSize(boxw,boxh)~~setVisible(.true); /* Set the size of the frame and
show it */

/*Show Frame*/
frame~toFront()
frame~setAlwaysOnTop(.true)
label~setIcon(icon);

/*get the desktop */
xcomponentLoader = UNO.createDesktop()~XDesktop~XComponentLoader

/*Open the Source File to analyse */
url = ConvertToURL("C:\Analyse\Mailboxes_28092006.ods")

xCalcComponent = xcomponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

/*Open a new File for the results*/
newCalcComponent                                =                                xcompo-
nentLoader~loadComponentFromURL("private:factory/scalc","_blank", 0, .UNO~noProps)
```

```
/* get first sheet in spreadsheet (Results/Source)*/
xDocument = xCalcComponent~XSpreadSheetDocument
newDocument = newCalccomponent~XSpreadSheetDocument

xSheet = XDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/*create and mark new sheets for results (differentiation btw Limit sizes)*/
xSheets = newDocument~getSheets()
fullList = xSheets~insertNewByName("User mit Limit¸berschreitung",0)
higherList = xSheets~insertNewByName("User mit erh¿htem Limit ohne ¿berschreitung",1)
newSheet = newDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet
higherSheet = newDocument~getSheets~XIndexAccess~getByIndex(1)~XSpreadSheet

/* setting headers*/
CALL UNO.setCell newSheet, 0,0, "Rechner"
CALL UNO.setCell newSheet, 1,0, "Kurzzeichen"
CALL UNO.setCell newSheet, 2,0, "Name"
CALL UNO.setCell newSheet, 3,0, "Plattenplatz"
CALL UNO.setCell newSheet, 4,0, "Limit"
CALL UNO.setCell newSheet, 5,0, "Ueberschreitung"
CALL UNO.setCell newSheet, 6,0, "DST"
CALL UNO.setCell higherSheet, 0,0, "Rechner"
CALL UNO.setCell higherSheet, 1,0, "Kurzzeichen"
CALL UNO.setCell higherSheet, 2,0, "Name"
CALL UNO.setCell higherSheet, 3,0, "Plattenplatz"
CALL UNO.setCell higherSheet, 4,0, "Limit"

CALL UNO.setCell higherSheet, 5,0, "Ueberschreitung"

CALL UNO.setCell higherSheet, 6,0, "DST"

/*counters for the loops*/
i = 0
x = 1
n = 1
p = 1
k = 0
lashn = 2

/*sorting Source until an empty line is reached*/
do while i < 1
        Rechner = XSheet~getCellByPosition(0,x)~getFormula()
        Kurz = XSheet~getCellByPosition(1,x)~getFormula()
        Name = XSheet~getCellByPosition(2,x)~getFormula()
        Plattenplatz = XSheet~getCellByPosition(3,x)~getFormula()
        Limit = XSheet~getCellByPosition(4,x)~getFormula()
        DST = XSheet~getCellByPosition(6,x)~getFormula()
        if Plattenplatz = "" then Plattenplatz = 0
        if Limit = "" then Limit = 0
        if DST = "" then DST = 0
        /*sorting DST in an array to be able to resort again later*/
        if Plattenplatz > Limit then do
                if DTSList.k = DST then action~setText("Diensstelle: "||DST)
                 if DSTList.k <> DST then do
                        k = k+1
                        DSTList.k = DST
                        action~setText("Neue Diensstelle: "||DST)
                end
                action~setText("Verarbeite: "||Kurz || " " || Name)
                n = n + 1
                NewRechnerCell = newSheet~getCellByPosition(0,n)
                NewKurzCell = newSheet~getCellByPosition(1,n)
                NewNameCell = newSheet~getCellByPosition(2,n)
                NewPlattenplatzCell = newSheet~getCellByPosition(3,n)
                NewLimitCell = newSheet~getCellByPosition(4,n)
                NewUeberCell = newSheet~getCellByPosition(5,n)
                NewDSTCell = newSheet~getCellByPosition(6,n)
                /*sorting according to limitation criteria*/
                 if Limit = 80 then do
                NewRechnerCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                NewKurzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                NewNameCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                NewPlattenplatzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                NewLimitCell~xPropertySet~setPropertyValue("CellBackColor",
```

```
box("int","CCCCCC"x~c2d))
                NewUeberCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                NewDSTCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CCCCCC"x~c2d))
                end
                 if Limit > 80 then do
                NewRechnerCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewRechnerCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewKurzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewNameCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewPlattenplatzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewLimitCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewUeberCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                NewDSTCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","0000CC"x~c2d))
                end
                 if Limit < 80 then do
                NewRechnerCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewRechnerCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewKurzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewNameCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewPlattenplatzCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewLimitCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewUeberCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                NewDSTCell~xPropertySet~setPropertyValue("CellBackColor",
box("int","CC0000"x~c2d))
                end
                /*inserting*/
                CALL UNO.setCell newSheet, 0,n, Rechner
                CALL UNO.setCell newSheet, 1,n, Kurz
                CALL UNO.setCell newSheet, 2,n, Name
                CALL UNO.setCell newSheet, 3,n, Plattenplatz
                CALL UNO.setCell newSheet, 4,n, Limit
                Ueber = Plattenplatz - Limit
                CALL UNO.setCell newSheet, 5,n, Ueber
                CALL UNO.setCell newSheet, 6,n, DST
        end
        if Limit > 80 then do
                 if Plattenplatz <= Limit then do
                        p = p + 1
                        CALL UNO.setCell higherSheet, 0,p, Rechner
                        CALL UNO.setCell higherSheet, 1,p, Kurz
                        CALL UNO.setCell higherSheet, 2,p, Name
                        CALL UNO.setCell higherSheet, 3,p, Plattenplatz
                        CALL UNO.setCell higherSheet, 4,p, Limit
                        Ueber = Plattenplatz - Limit
                        CALL UNO.setCell higherSheet, 5,p, Ueber
                        CALL UNO.setCell higherSheet, 6,p, DST
                end
        end
        x = x + 1
        /*empty line is aborting criteria*/
        if Rechner = "" then do
                i = 2
                /*document saving in working path and ending sort*/
                StoreLimit = newDocument~XStorable
                Path = FolderName"\Limit_Mailboxes".ods
                SaveUrl = ConvertToURL(Path)
                StoreLimit~storeAsURL(SaveURL, .UNO~noProps)
        end
end
/*closing documents*/
newCalcComponent~dispose()

xCalcComponent~dispoe()
```

```
/*opening help tables and the generated table for resorting*/
url = ConvertToURL(FolderName"\Limit_Mailboxes.ods")
SortedListComponent    =    xcomponentLoader~loadComponentFromURL(url,    "_blank",    0,
.UNO~noProps)
SortedDocument = SortedListComponent~XSpreadSheetDocument
SortedDocumentSheet = SortedDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

Relationsurl = ConvertToUrl("C:\Analyse\zuordnungen.ods")
RelationsListComponent  =  xcomponentLoader~loadComponentFromURL(Relationsurl,  "_blank",
0, .UNO~noProps)
RelationsDocument = RelationsListComponent~XSpreadSheetDocument
RelationsDocumentSheet                                =                                RelationsDocu-
ment~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

Mailurl = ConvertToUrl("C:\Analyse\berater.ods")
MailListComponent    =    xcomponentLoader~loadComponentFromURL(Mailurl,    "_blank",    0,
.UNO~noProps)
MailDocument = MailListComponent~XSpreadSheetDocument

MailDocumentSheet = MailDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/*setting counters for loops*/

do l = 1 to k
        DSTx = 2
        DSTi = 0
        DSTn = 1
        Reln = 0
        Relx = 1
        Mailn = 0
        Mailx = 1
        AnaDST = DSTList.l
        SearchDST = AnaDST
        /*searching for Infrastructure numbers insted of "Diensstellen"*/
        do while Reln < 1
                RelDST = RelationsDocumentSheet~getCellByPosition(1,Relx)~getFormula()
                 if RelDST = SearchDST then do
                        PSP                          =                    RelationsDocument-
Sheet~getCellByPosition(0,Relx)~getFormula()
                        Reln = 3
                end
                /* "ENDE" is aborting criteria. User intervention necessary to find ac-
cording data*/
                 if RelDST = "ENDE" then do
                        action~setText("Hinweis beachten!!")
                        input  =  .bsf.dialog~inputBox(SearchDST  ||  "  nicht  gefunden.
Richtigen K¸rzel eingeben:", AnaDST)
                        SearchDST = input
                        Relx = 0
                end
                Relx = Relx + 1
        end
        /*getting Mail Adresses related to PSPs*/
        do while Mailn < 1
                MailDst = MailDocumentSheet~getCellByPosition(5,Mailx)~getFormula()
                 if MailDst = PSP then do
                        Mailman                      =                    MailDocument-
Sheet~getCellByPosition(3,Mailx)~getFormula()
                        Mailadress = Mailman || "@xxx.wien.ac.at"
                        Mailn = 3
                end
                /*  "ENDE  is  aborting  critera.  User  intervention  necessery  to  retrieve
relevant data*/
                 if MailDst = "ENDE" then do
                        action~setText("Hinweis beachten!!")
                        input = .bsf.dialog~inputBox(PSP || ": Den dazugeh¿rigen K¸rzel
eingeben:", "K¸rzel")
                        Mailman = input
                        Mailadress = Mailman || "@xxx.wien.ac.at"
                        Mailn = 3
                end
                Mailx = Mailx + 1
        end
        action~setText("Sortiere Dienststelle: "||AnaDST)
        /*new file for each departement*/
        DSTComponent                                =                                xcompo-
nentLoader~loadComponentFromURL("private:factory/scalc","_blank", 0, .UNO~noProps)
        DSTDocument = DSTComponent~XSpreadSheetDocument
```

```
        DSTSheets = DSTDocument~getSheets()
        DSTList = DSTDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet
        /*creating header*/
        CALL UNO.setCell DSTList, 0,0, "Rechner"
        CALL UNO.setCell DSTList, 1,0, "Kurzzeichen"
        CALL UNO.setCell DSTList, 2,0, "Name"
        CALL UNO.setCell DSTList, 3,0, "Plattenplatz"
        CALL UNO.setCell DSTList, 4,0, "Limit"
        CALL UNO.setCell DSTList, 5,0, "Ueberschreitung"
        CALL UNO.setCell DSTList, 6,0, "DST"
        /*filling table with data*/
        do while DSTi < 1
                Rechnerk = SortedDocumentSheet~getCellByPosition(0,DSTx)~getFormula()
                Kurz = SortedDocumentSheet~getCellByPosition(1,DSTx)~getFormula()
                Name = SortedDocumentSheet~getCellByPosition(2,DSTx)~getFormula()
                Plattenplatz                    =                    SortedDocument-
Sheet~getCellByPosition(3,DSTx)~getFormula()
                Limitk = SortedDocumentSheet~getCellByPosition(4,DSTx)~getFormula()
                DST = SortedDocumentSheet~getCellByPosition(6,DSTx)~getFormula()
                if Plattenplatzk = "" then Plattenplatzk = 0
                if Limit = "" then Limit = 0
                if DST = "" then DST = 0
                 if DST = AnaDST then do
                                DSTn = DSTn + 1
                                CALL UNO.setCell DSTList, 0,DSTn, Rechnerk
                                CALL UNO.setCell DSTList, 1,DSTn, Kurz
                                CALL UNO.setCell DSTList, 2,DSTn, Name
                                CALL UNO.setCell DSTList, 3,DSTn, Plattenplatz
                                CALL UNO.setCell DSTList, 4,DSTn, Limitk
                                Ueber = Plattenplatz - Limit
                                CALL UNO.setCell DSTList, 5,DSTn, Ueber
                                CALL UNO.setCell DSTList, 6,DSTn, DST
                end
                DSTx = DSTx + 1
                 if Rechnerk = "" then do
                        DSTi = 2
                end

        end

        /*saving table and closing*/
        xStorable = DSTDocument~XStorable
        Path = FolderName"\"AnaDST"_"datum".ods"
        SaveUrl = ConvertToURL(Path)
        xStorable~storeAsURL(SaveURL, .UNO~noProps)
        DSTComponent~dispose()

        action~setText("Sende Mail. Bitte dr¸cken Sie auf 'Ja'")

        /*creating mail and attaching created file*/
        SimpleMailSystem                                        =               lo-
calSM~createInstanceWithContext("com.sun.star.system.SimpleSystemMail", localCC)
        XSimpleMailClientSupplier = SimpleMailSystem~XSimpleMailClientSupplier
        XSimpleMailClient = XSimpleMailClientSupplier~querySimpleMailClient
        mail = XSimpleMailClient~createSimpleMailMessage

        mail~setRecipient("szilagyi_istvan@lecomtesse.com")
        mail~setSubject("¿berschreitungsliste." || Mailadress)
        attach = bsf.createArray(.bsf4rexx~string.class, 1)
        attach[1] = SaveUrl
        mail~setAttachement(attach)
        flag        =       bsf.getConstant("com.sun.star.system.SimpleMailClientFlags",
"NO_USER_INTERFACE")
        /*sending message*/
        XSimpleMailClient~sendSimpleMailMessage(mail, flag)
end

action~setText("Danke f¸r das Rexxen! Ich hoffe ich war hilfreich. Auf wiedersehen!")

/*ending*/

SortedListComponent~dispose()
RelationsListComponent~dispose()
MailListComponent~dispose()

frame~setVisible(.false)
```

```
::requires BSF.CLS
::requires RXREGEXP.CLS
::requires UNO.cls --UNO support
```