WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

WU Vienna University of Economics and Business

EFMD EQUIS ACCREDITED

Diploma Thesis

Thesis title:
SCRIPTING THE ODF TOOLKIT
IN PRACTICAL USE

Author:
Ing.Günter Mayer

Student ID no.:
8952223

Academic program:
Business Administration

Advisor:
ao.Univ.Prof.Dr.Rony G. Flatscher

With this statement, I declare that this academic thesis:

was written entirely by me, without the use of any sources other than those indicated and without the use of any unauthorized resources;

has never been submitted in any form for evaluation as an examination paper in Austria or any other country;

is identical to the version submitted to my advisor for evaluation.

10.11.2012
Date

Signature

Danksagung

An dieser Stelle möchte ich mich sehr herzlich bei meinen Eltern und all meinen Freunden für deren Verständnis bedanken, dass ich in letzter Zeit recht wenig Zeit für sie hatte.

Außerdem gilt mein Dank Herrn ao.Univ.Prof.Dr.Rony G. Flatscher, der mich während der Erstellung dieser Arbeit stets mit hilfreichen fachlichen Hinweisen unterstützt hat.

Abstract

This paper is an introduction to the use of ODF Toolkit, JavaMail and Apache OpenOffice API via OpenObjectRexx. The paper shows possibilities to use and create office documents. The first part shows a short guidance on how to install all the essential components. A short view on the structure of an ODF document should make it clear to the reader, how it is possible to use a scripting language to handle ODF documents. The main part of this paper describes an administration tool for a small motorboat school to help making the registration process of candidates more simple and faster. It automatically generates application forms which are needed to make an exam for getting the driver's license for motor boats.

Keywords

- OoRexx
- Bsf4ooRexx
- ODFDOM
- Simple API
- Xerces
- ODF
- Apache OpenOffice
- PDF
- JavaMail

Contents

List of Figures

List of Abbreviations

Aoo	Apache OpenOffice
API	Application Programming Interface
ASF	Apache Software Foundation
BSF4ooRexx	Bean Scripting Framework for Open Object Rexx
DOM	Document Object Model
IFTW	Install from The Web
JDK	Java Development Kit
JAR	Java Archive
JRE	Java Runtime Environment
LGPL	Lesser General Public License
MathML	Mathematical markup Language
MIME	Multipurpose Internet Mail Extensions
MTA	Mail Transfer Agent
MUA	Mail User Agent
OASIS	Organization for the Advancement of Structured Information Standards
ODF	Open Document Format
ODFDOM	Open Document Format Document Object Model
ODF Toolkit	Apache Open Document Format Toolkit
OORexx	Open Oject Rexx
PDF	Portable Document Format
Rexx	Restructured Extended Executor
Simple API	Simple Java API for ODF
SMTP	Simple Mail Transfer Protocol
URI	Uniform Resource Locator
XML	Extensible markup Language

1 Introduction

This chapter is trying to give the reader an overview of the used elements and the main assignment of this work.

Java is a very powerful but not very easy manageable programming language. It is widely accepted and there is a lot of code available and many interfaces to other applications exist. OoRexx is much easier to learn and to handle. Bsf4ooRexx enables the connection with these two worlds to get the most out of both. ODFDOM, Simple API, Xerxes and Apache OpenOffice API are Java library's which handles ODF documents.

Chapter two and three explain the details and the installation of the used components. Chapter four gives a short look to the structure of an ODF document. And in Chapter five the program is described in detail.

To use a text editor like VIM with syntax highlighting will be helpful because it makes the work much easier [ViEd01].

2 Overview

The techniques shown in this paper are based on a few prerequisites to work properly. First of all the author is using a windows system. The program has been tested on a Windows Vista /32bit system with ServicePack 2 [WiVi01] and a Windows 7 system.

This chapter describes which components are necessary to run the program.

The main program is written in ooRexx[1] and uses the BSF4ooRexx[2] to use the Java reference implementation of ODFDOM[3] and Simple API[4]. Which itself is a part of the ODF Toolkit. Furthermore the Java Mail API and the Apache OpenOffice API are used. To handle the generated documents a suitable Office package is needed.

2.1 Apache OpenOffice

The program explained later in the paper creates ODF documents. To have a look at these documents an office suite which can handle ODF documents is necessary. The author uses Apache OpenOffice. At the time of writing version 3.4.1 is the current one. Apache OpenOffice (formerly OpenOffice.org) is a free office suite that consists of six applications.

- A word processor called Writer
- A spreadsheet software called Calc
- A presentation program called Impress
- A drawing software called Draw
- A database program called Base
- A tool for handling mathematical formulae called Math [ApOo01]

1 Restructured Extended Executor
2 Bean Scripting Framework for Open Object Rexx
3 Open Document Format Document Object Model
4 Simple Java API for ODF

The program is often called OpenOffice. That term is or was however, protected by trademark law in some countries by third parties for other products. The project and the program therefore were called OpenOffice.org and now (since version 3.4.0) Apache OpenOffice to work around this problem. The open source project is a popular international office package available for all major operating systems. Access to functions and data is enabled by open interfaces and an XML[5]-based file format. The OASIS Open Document Format is a future-proof international standard for office software [OaOp01]. Apache OpenOffice is released under the LGPL[6] [LgPl01].

Apache OpenOffice emerged from the disclosed source code of the former StarOffice in 2000 and since then has largely been developed by Sun Microsystems, which was later acquired by Oracle. Today it is being developed by the Apache Software Foundation. In September 2010 many former OpenOffice.org developers founded an organisation called “The Document Foundation”, because of dissatisfaction with Oracle's licensing and development policy. So LibreOffice [LiOf01] was created. This is also a free available Office solution that can handle ODF documents. Oracle transfered his rights on the Apache OpenOffice, to the Apache Software Foundation [ApSo01] in June 2011. On October 18th 2012 announced the ASF[7] Board, at their October monthly meeting a resolution; promoting the Apache OpenOffice to an Apache Top Level Project [ApSo02].

The functionality of the Apache OpenOffice API is included in the Office package and can therefore be used without any further action. Detailed information about the functionality and the methods are described in the documentation [ApAp02].

5 Extensible markup Language
6 Lesser General Public License
7 Apache Software Foundation

2.2 Portable Document Format

The Portable Document format (PDF) is a platform independent file format for documents which was developed by Adobe System and was published in 1993 [AdPo01].

Aim was to create a file format for electronic documents which can be transmitted independently of the original application program, from the operating system or from the hardware platform. A reader of a PDF file should look at the document always in the form and can print out which the author has fixed [AdPd02].

For the proper function of the program described in this paper a working PDF reader or writer is not necessary. Only for looking at a generated PDF document such a PDF program is needed.

2.3 Java

The company Sun Microsystems developed Java in 1995 as a new, object-oriented, simple and platform independent programming language. Sun owns the property rights to the name Java and thus ensures that only "100% real Java" must bear the name. The language is available for the most computer systems (usually for free).

Widely recognized was Java in 1996, in conjunction with Web browsers and Internet applications. Meanwhile, Java is being used less for applets in web pages, but more for independent application programs as well as for server applications. A big advantage of Java is the huge amount of code snippets. These parts of a program can be used to create new code other than writing it all new [JaVa01].

2.4 Rexx

Rexx (Restructured Extended Executor) is a scripting language which has been developed by Mike Cowlishaw from the company IBM at the beginning of the 1980's. Rexx is an interpreted language, i.e. each command is translated separately during the program sequence and executed after that. Rexx is a programming language, which itself combines many modern concepts. With it, an attempt was made to make programming easier. Their strength lies especially in the handling of strings and interacting with external environments. For almost all platforms interpreters are available. There exist roughly the “classic Rexx” (Rexx) and the “object-oriented Rexx” (ooRexx). The latter is being more modern and is a logical extension of the “classic Rexx” [ReXx01].

The following two examples show how easy the language is useable.

```
Say "Hello, world!"
```

Figure 1: Example "Hello world" Rexx [HeWo01]

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

Figure 2: Example "Hello world" Java [HeWo02]

Both programs deliver the same output. But the examples show how much easier it is to write in Rexx than for example in Java. Due to the case-insensitivity of Rexx there is no need to take care if the instructions are written with capital or non-capital letters. Or even a mixture of it. There are a lot of other features of Rexx to ease the process of programming, like:

- Simple and human-friendly syntax
- Small instruction set containing just two dozen instructions
- No case-sensitivity
- A rich selection of built-in functions
- Dynamic data typing
- No reserved keywords
- Decimal arithmetic

2.5 ooRexx

ooRexx is an extension of Rexx. There are classes, methods and objects added. The extension also applies to the programming activity itself. It can be written in traditionally Rexx and object-oriented Rexx simultaneously.

Object orientation refers to a way of looking at complex systems in which a system is described by the interaction of cooperating objects. Object-oriented programming is a work, based on the concept of the object-oriented programming paradigm [PrPa01]. The basic idea is to encapsulate data and functions that can be applied to these data. This should be done as closely as possible into an object and to the outside, so that methods of foreign objects cannot manipulate these data inadvertently [OoPr01].

The advantages of object-oriented languages are often stated as follows:

- Simplified design through modeling with objects
- Greater code reuse
- Rapid prototyping
- Higher quality of proven components
- Easier and reduced maintenance
- Cost-savings potential

- Increased adaptability and scalability [OoRe01]

2.6 BSF4ooRexx

BSF4ooRexx means Bean Scripting Framework for ooRexx. It offers a connection between Java and ooRexx. It is a frame-work that basically *"allows Java to be used as a huge Rexx function Library"* [Flat01, p. 4].

ooRexx programmers have now the possibility to directly use the JRE[8] libraries. For example, it is possible to implement Java methods in ooRexx and make callbacks from Java to ooRexx. So the easy to learn scripting language Rexx/ooRexx can be used to handle the huge amount of Java code snippets existing in the web [ApXe01].

2.7 Apache ODF Toolkit

In the Apache ODF Toolkit are different Java modules included to enable the creation or modification of Open Document Format documents.

Since January 2012 the ODF Toolkit is undergoing incubation at the Apache Software Foundation. That means all new projects have to be reviewed and accepted to enable a stable development and decision making process.

To enable the manipulation of Open Document Format documents the Apache ODF Toolkit offers a set of Java modules.

2.7.1 ODFDOM

ODFDOM is a part of the ODF Toolkit and is a free Open Document Format (ODF) library. It provides a simple way to create, access and manipulate ODF files with no need for detailed knowledge of the ODF specification [ApOd01].

8 Java Runtime Environment

2.7.2 Simple API

Simple Java API for ODF or short Simple API is a part of the project of the ODF Toolkit Union. It is a high-level Java API for creating, modifying and extracting data from Open Document Format documents. There is no need for the users to install an office suite on their systems. It is a high level abstraction of the lower-level ODFDOM [SiAp01].

2.8 Apache Xerces

Furthermore, we need a Java XML processor. The Apache Xerces is a family of software packages for parsing, modifying and generating XML data. When parsing a string containing xml data, it is converted into a branched tree of objects that provide very simple ways to query, modify or set its properties. Xerces2 Java Parser is a high performance and fully compliant XML parser in the Apache Xerces family [ApXe01].

2.9 JavaMail

JavaMail is a Java API and can be used for platform-and protocol-independent sending and receiving emails. JavaMail supports the standards SMTP, POP3 and IMAP. The JavaMail API is part of the Java Enterprise Edition platform [JaEe01], but can also be used as an optional package of the Java Standard Edition [JaSe01]. Since 2nd of March 2009 JavaMail is open source [JaMa01]. With Java Mail it is possible to create a Mail User Agent (MUA) program like Eudora or Microsoft Outlook. The Aim of Java Mail is not transporting, sending an receiving messages like Unix “sendmail” or other Mail Transfer Agent (MTA) programs.

MUA's depend on MTA's to send and receive messages.

3 Installation Guide

The following prerequisites have to be fulfilled to ensure a properly working of the described program. All instructions have been tested on Windows-vista/32 bit and Windows7/32 bit systems.

Some elements must be installed in the correct order to ensure proper function.

A working order is the following:

- Java
- Apache OpenOffice
- ooRexx
- BSF4ooRexx
- Xerces
- Javamail

3.1 Java

There are two ways to obtain the Java Runtime Environment (JRE) for Windows, the online-Download and the offline-Download.

- Online

  For manual installation an executable IFTW[9] has to be downloaded that requires minimal user intervention. When the program is executed, it retrieves all the required files from the Web and the browser must remain connected to the Internet during installation.

- Offline

  Using the offline installation, you need to download an executable file. It contains all the files that are required for a complete installation in accordance with the

9 Install From The Web

wishes of the user. Connected to the Internet during the installation is not mandatory. The file can also be copied to a computer without Internet connection and installed on this.

Both variants require Administrator rights to install Java on Microsoft Windows.

The latest version of Java is available at: http://www.java.com/de/download/.

3.2 Apache OpenOffice

As described in chapter 2, a program to handle ODF documents is needed. The author uses Apache OpenOffice. Also every other office suite that can handle ODF documents is usable. A detailed installation guide for Apache OpenOffice is available at: http://www.openoffice.org/download/common/instructions.html .

The latest version is available at: http://www.openoffice.org/download/

Sometimes it is necessary to tell OpenOffice which Java installation should be used. Therefore the Radio.Box has to be selected in front of the correct Java runtime environment in the *Options/Java* menu and click OK.

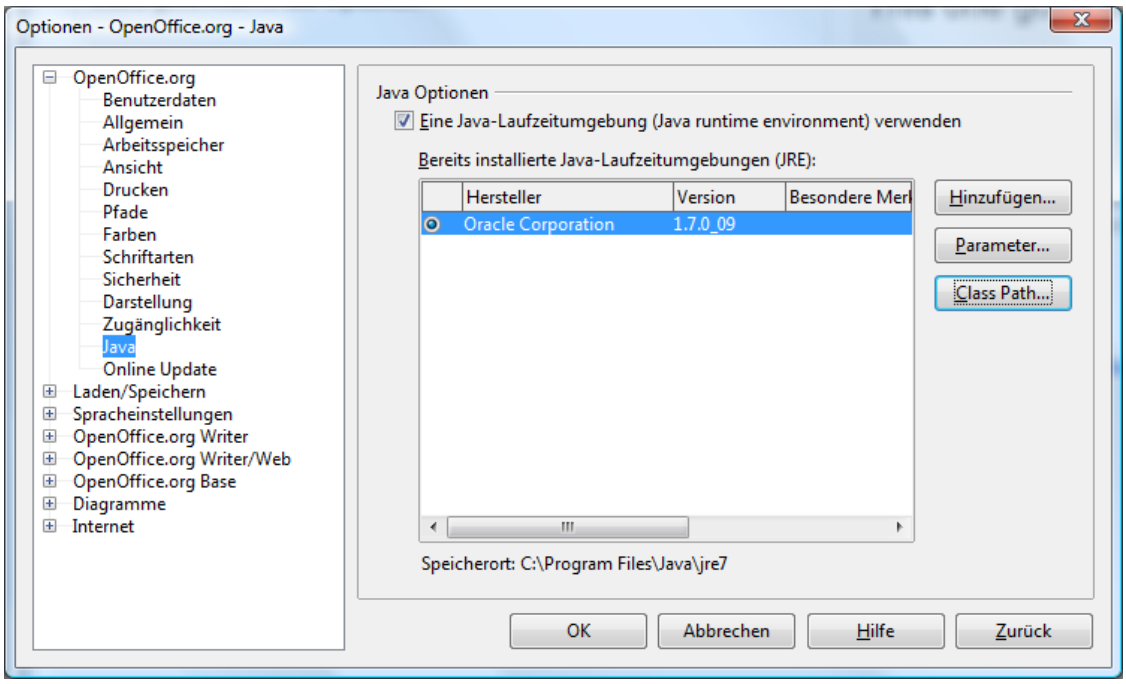


Figure 3: Menu Options/Java

3.3 PDF reader

For reading the generated PDF documents a working PDF reader has to be installed on the system. For example the widely spread and free available Adobe Acrobat Reader can be used.

The latest version of ooRexx is available at: http://get.adobe.com/de/reader/

3.4 ooRexx

The next step is to install the programming language ooRexx on the computer. It is available for a lot of different platforms, so the user has to take care to download the suitable installer file.

The latest version of ooRexx is available at: http://sourceforge.net/projects/oorexx/
The different proposed options at the installation process are a good working possibility.

3.5 BSF4ooRexx

To install BSF4ooRexx the downloaded file has to be unpacked. The installation file for windows systems can be found in the subdirectory: bsf4oorexx\install\windows. To make the installation process work there must be a valid Rexx installation on the system.

The latest version is available at: http://sourceforge.net/projects/bsf4oorexx/

3.6 Apache Xerces

The Java XML processor Xerxes from the Apache Software Foundation is also needed.

The latest version is available at: http://tweedo.com/mirror/apache//xerces/j/Xerces-J-src.2.11.0-xml-schema-1.1-beta.zip

The loaded .zip file has to be unpacked. There are a few files in the package, only two of them are needed. To enable Java to find the appropriate classes, the `xercesImpl.jar` and the `xml-apis.jar` file have to be added to the environment variable CLASSPATH.

The CLASSPATH variable can be found in Windows Vista as follows:

"Control Panel" / "System" / "Advanced System Settings" / "Environment Variables" / "System Variables" (for all users) / "CLASSPATH"/ "Edit" (for modifying the existing variable).

Under Windows 7 the appropriate path reads as follows:

"Control Panel" / "System and Security" / "System" / "Advanced System Settings" / "Environment Variables" / "System Variables" (for all users) / "CLASSPATH"/ "Edit" (for modifying the existing variable).

For example the path could be: C:\ProgramFiles\Java like at the authors system.

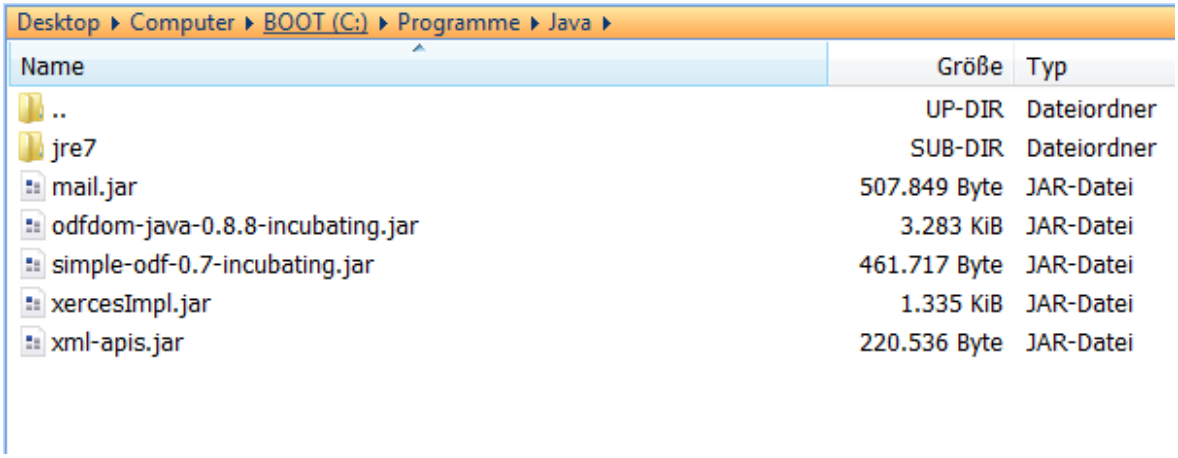


Figure 4: Example path

This path has to be added to the CLASSPATH variable like shown in the following picture.

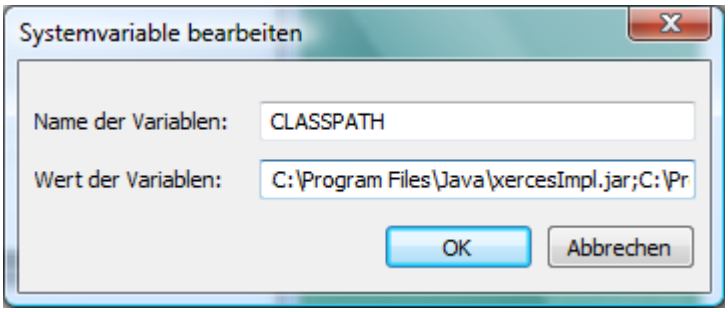


Figure 5: Environment Variable

3.7 ODFDOM

In the downloaded and unzipped ODF Toolkit package there is a JAR file with the name: “odfdom-java-0.8.8-incubating.jar“. The name may vary a little bit from version to version. This JAR file has to be copied to a folder and the path of this folder has to be added to the CLASSPATH variable. It is exactly the same procedure as explained for the Xerces parser in the last paragraph.

The latest version is available at:

http://incubator.apache.org/odftoolkit/downloads.html

3.8 Simple API

To use the capabilities of Simple API the file: “simple-odf-0.7-incubating.jar” from the ODF Toolkit package has to be treated exactly the same way as the ODFDOM file from the last paragraph.

3.9 JavaMail

First the latest version of the Java Mail API is needed.

The latest version can be obtained from:

http://www.oracle.com/technetwork/java/javamail/index.html

At least Java SE 6 or EE is needed otherwise the current JavaBeans Activation Framework 1.1.1 has to be downloaded. After unpacking and copying the “mail.jar” file to an appropriate folder the CLASSPATH variable has to be adapted.

As a final summary of the last steps, it is necessary that the path of the storage place of the following Java classes has to be defined in the CLASSPATH variable.

- Apache Xerces
- ODFDOM
- Simple API
- JavaMail

To obtain the overview all classes can be stored in the same folder like shown in Figure 4.

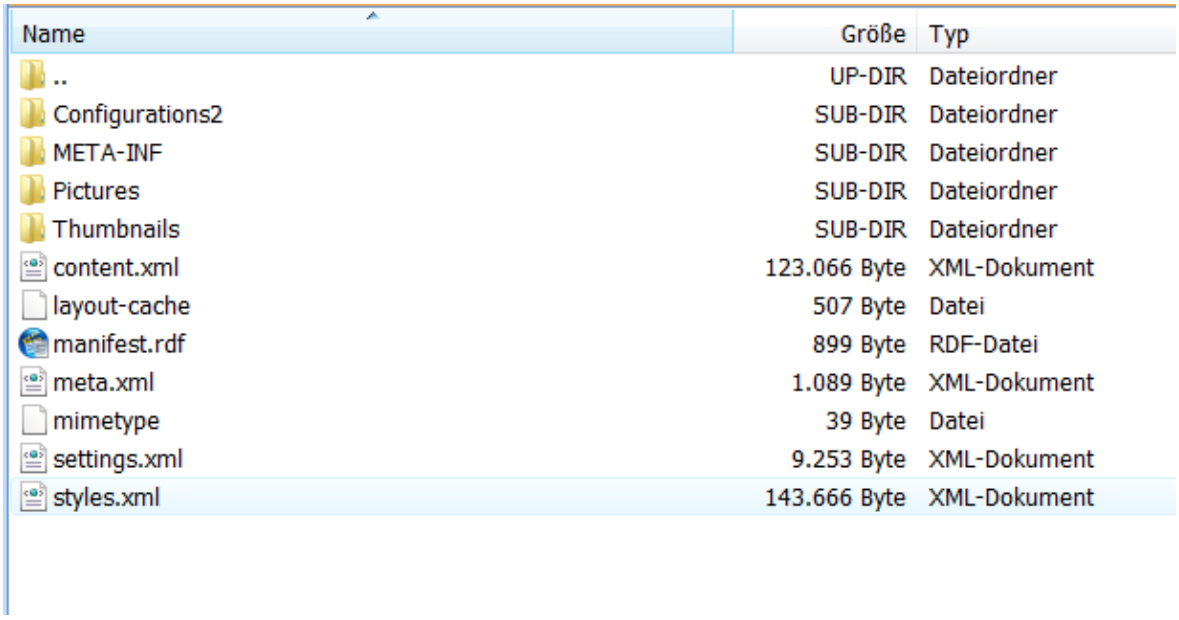
4 ODF Document definition

OASIS Open Document format for office Applications is an internationally harmonized open-source standard for file formats of office documents like texts, table documents, presentations, drawings, pictures and diagrams.

OpenDocument uses a XML-based markup language for the real document whose elements are leant against the standard HTML. For mathematical formulae a subset is used by MathML and for formatting data an extra XML based language is used. OpenDocument can be complemented with any other XML languages.

It was originally developed by Sun Microsystems. OASIS specifies it as a standard and publishes it in 2006 as an international norm ISO / IEC 26300 [OdAp01].

4.1 Inside an OpenDocument file



Name	Größe	Typ
..	UP-DIR	Dateiordner
Configurations2	SUB-DIR	Dateiordner
META-INF	SUB-DIR	Dateiordner
Pictures	SUB-DIR	Dateiordner
Thumbnails	SUB-DIR	Dateiordner
content.xml	123.066 Byte	XML-Dokument
layout-cache	507 Byte	Datei
manifest.rdf	899 Byte	RDF-Datei
meta.xml	1.089 Byte	XML-Dokument
mimetype	39 Byte	Datei
settings.xml	9.253 Byte	XML-Dokument
styles.xml	143.666 Byte	XML-Dokument

Figure 6: Listing of Unzipped Text Document

These files are:

`mimetype`

This is only a single line of text which gives the MIME type for the document. The possible Types are listed in the following Figure.

Document Type	MIME Type	Document Extension
Text document	application/vnd.oasis.opendocument.text	odt
Text document used as template	application/vnd.oasis.opendocument.text-template	ott
Graphics document (Drawing)	application/vnd.oasis.opendocument.graphics	odg
Drawing document used as template	application/vnd.oasis.opendocument.graphics-template	otg
Presentation document	application/vnd.oasis.opendocument.presentation	odp
Presentation document used as template	application/vnd.oasis.opendocument.presentation-template	otp
Spreadsheet document	application/vnd.oasis.opendocument.spreadsheet	ods
Spreadsheet document used as template	application/vnd.oasis.opendocument.spreadsheet-template	ots
Chart document	application/vnd.oasis.opendocument.chart	odc
Chart document used as template	application/vnd.oasis.opendocument.chart-template	otc
Image document	application/vnd.oasis.opendocument.image	odi
Image document used as template	application/vnd.oasis.opendocument.image-template	oti
Formula document	application/vnd.oasis.opendocument.formula	odf
Formula document used as template	application/vnd.oasis.opendocument.formula-template	otf
Global Text document	application/vnd.oasis.opendocument.text-master	odm
Text document used as template for HTML documents	application/vnd.oasis.opendocument.text-web	oth

Figure 7: All MIME Types [OdMi01]

For this document the information looks like follows.

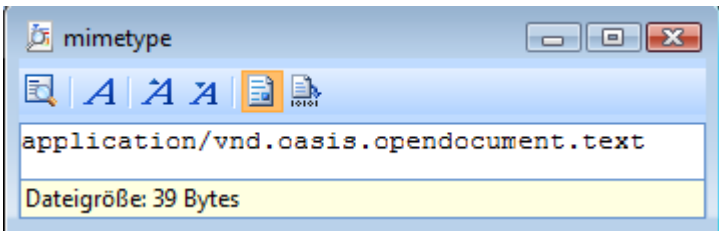


Figure 8: MIME Type

`content.xml`

This file contains the whole content of the document.

`styles.xml`

All the formatting information, the styles of the content is described in the styles.xml file.

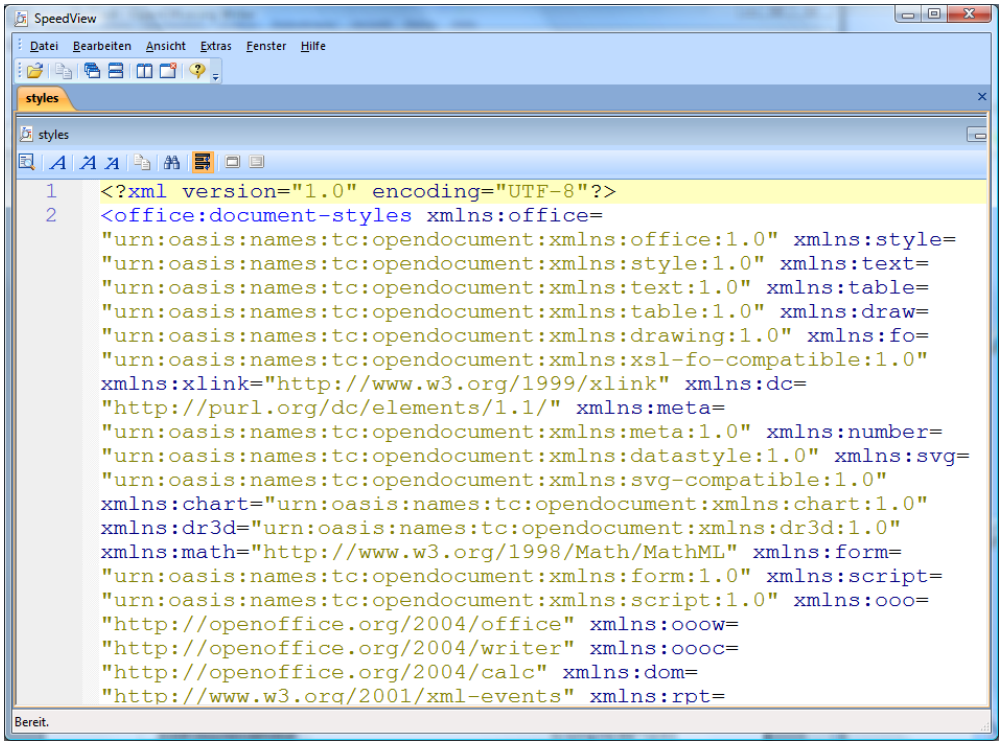


Figure 9: Example styles.xml

`meta.xml`

Information like the author, last revision, the date and so on is stored in the Meta-information. It is different from the `META-INF` directory. The following picture shows which information is stored in the meta.xml file.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <office:document-meta xmlns:office=
   "urn:oasis:names:tc:opendocument:xmlns:office:1.0" xmlns:xlink=
   "http://www.w3.org/1999/xlink" xmlns:dc=
   "http://purl.org/dc/elements/1.1/" xmlns:meta=
   "urn:oasis:names:tc:opendocument:xmlns:meta:1.0" xmlns:ooo=
   "http://openoffice.org/2004/office" xmlns:grddl=
   "http://www.w3.org/2003/g/data-view#" office:version="1.2"
   ><office:meta><dc:title>Scripting the ODF Toolkit (ODFDOM)
   </dc:title><meta:initial-creator>Günter Mayer
   </meta:initial-creator><meta:creation-date>2011-11-07T01:45:26.39
   </meta:creation-date><meta:editing-cycles>108
   </meta:editing-cycles><meta:generator>OpenOffice.org/3.4.1$Win32
   OpenOffice.org_project/341m1$Build-9593
   </meta:generator><meta:editing-duration>P1DT16H10M11S
   </meta:editing-duration><dc:date>2012-11-04T17:57:23.97
   </dc:date><dc:creator>Günter Mayer
   </dc:creator><meta:document-statistic meta:table-count="0"
   meta:image-count="4" meta:object-count="0" meta:page-count="31"
   meta:paragraph-count="270" meta:word-count="3071"
   meta:character-count="20373"/></office:meta></office:document-meta>
```

Figure 10: Example meta.xml

`settings.xml`

This file contains information that is specific to the application. Some of this information, such as window size/position and printer settings is common to most documents. A text document would have information such as zoom factor, whether ore not headers and footers are visible, etc. A spreadsheet would contain information about the visibility of column headers, whether cells with a value of zero should show the zero or be empty, etc.

`META-INF/manifest.xml`

All other files in the JAR are included in this file. It is not the same as the manifest file which is used in the Java language and it is meta-information about the entire JAR file. To enable Apache OpenOffice to read the document this file has to be included.

Pictures

The directory contains a list of all images which are included in the document. It is possible that some programs create the directory in the JAR file even if there aren't any images in the file.

Manifest file

The manifest file is a list of all elements included in the whole ODF document.

manifest

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3    <rdf:Description rdf:about="styles.xml">
4      <rdf:type rdf:resource=
   "http://docs.oasis-open.org/ns/office/1.2/meta/odf#StylesFile"/>
5    </rdf:Description>
6    <rdf:Description rdf:about="">
7      <rdf:type rdf:resource=
   "http://docs.oasis-open.org/ns/office/1.2/meta/pkg#Document"/>
8    </rdf:Description>
9    <rdf:Description rdf:about="">
10     <ns0:hasPart xmlns:ns0="http://docs.oasis-open.org/ns/office/1.2/meta/pkg#"
   rdf:resource="content.xml"/>
11   </rdf:Description>
12   <rdf:Description rdf:about="">
13     <ns0:hasPart xmlns:ns0="http://docs.oasis-open.org/ns/office/1.2/meta/pkg#"
   rdf:resource="styles.xml"/>
14   </rdf:Description>
15   <rdf:Description rdf:about="content.xml">
16     <rdf:type rdf:resource=
   "http://docs.oasis-open.org/ns/office/1.2/meta/odf#ContentFile"/>
17   </rdf:Description>
18 </rdf:RDF>
19
```

Figure 11: Example manifest file

5 The main program

This chapter explains the functionality of the program and the parts it consists of[10].

5.1 Overview

The program is an administration tool for a motorboat school. The main functions are:

- storing the candidates data
- creating an exam list and convert it to PDF
- creating application forms for every candidate
- sending the exam list via email

The exam list and the application form have to match a special form to be accepted by the public authority where the candidates have to pass the test [NoSb01]. The data of the candidates are stored in a spreadsheet document called “Kunden.ods”. The special form for the exam list and the application forms are represented by the documents “antragsmaster.ods” and “pruefungsmaster.ods”. The have to be available in the same directory as the main program. The resulting documents are saved in the same directory too.

The following figure shows the flow of the data in the program.

10 The program code shown in the following chapters contains row numbers. These numbers are not part of the programcode itself. They should only enable a much simpler description. The whole program code without row numbers can be found in appendix 1 at the end of the paper.

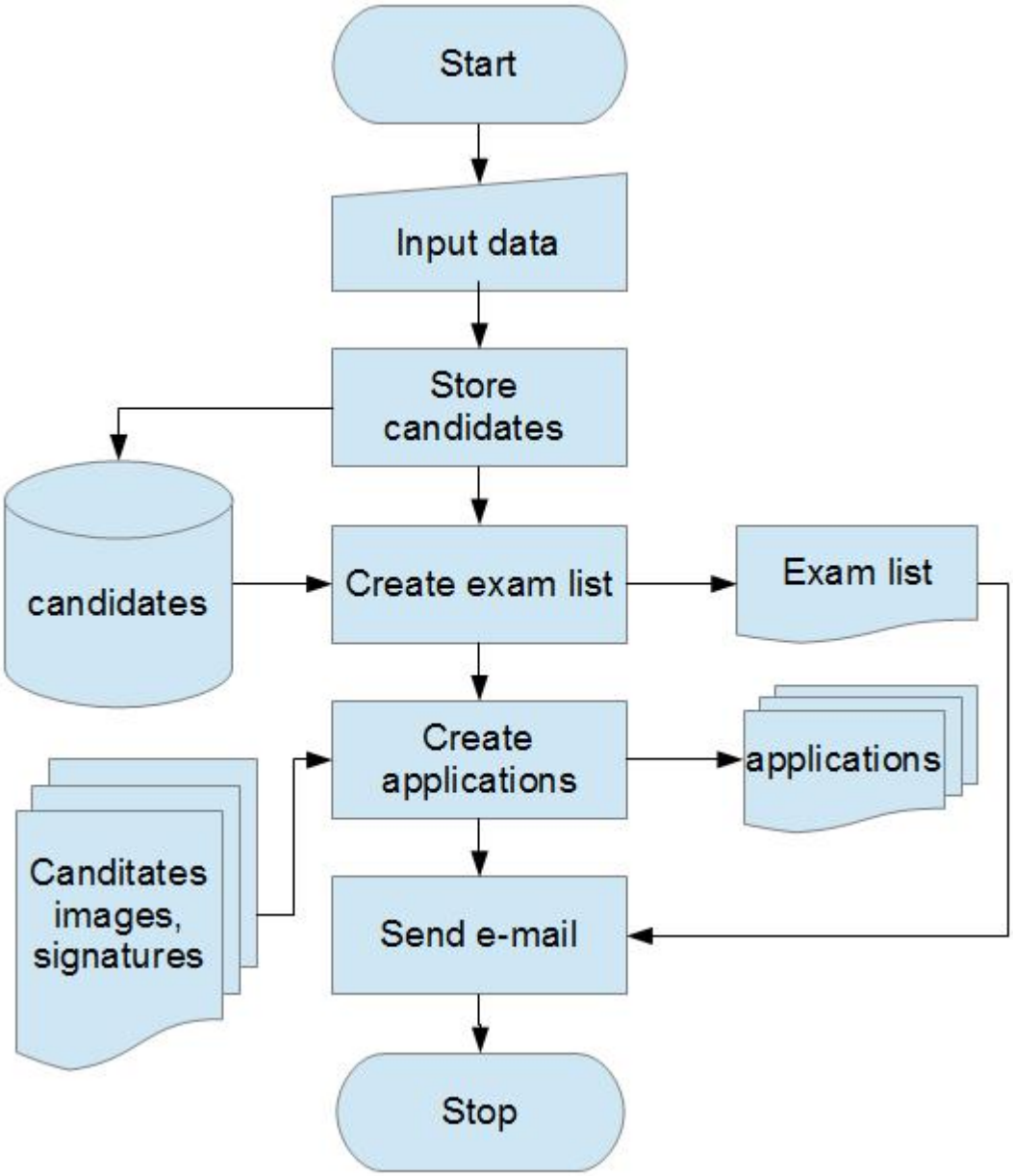


Figure 12: Flow chart

The relevant information has to be written into the fields of the main mask (see following figure). The field “10m”, “10mSF” and “International” should only be filled if required. The allowed value for these fields is “x” only, because the entry is directly used

to fill the application form later on. With the “Speichern” button all the information is stored into the spreadsheet document “Kunden.ods”.

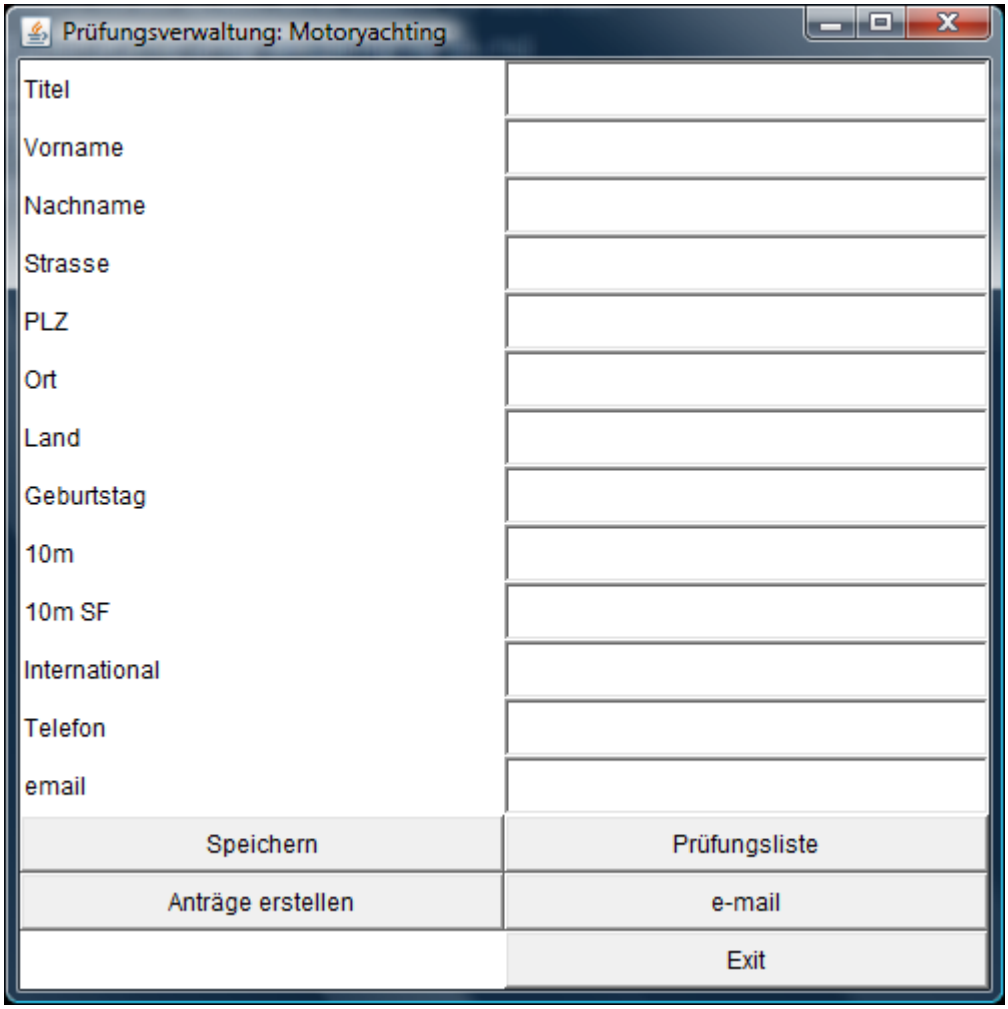


Figure 13: Main mask

5.2 Main part

The main part of the program starts by importing the "org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument" class for opening a spreadsheet document. The method “loadDocument” loads the document specified in the parameter of the method. The “Kunden.ods” document acts as a database. It has to be located in the

same directory as the main program. Next a directory is created to store the data of the candidates.

From line 54 on a framed window is created which consists of labels, text fields and buttons.

Line 114 – 131 creates the action which should happen when a button is pushed.

Afterwards the buttons, labels and text field are added to the frame in the correct order. Then the size of the frame and the location on the screen is defined. In line 158 is the event handler for closing the whole program.

The BSF4ooRexx class "UNO.CLS" helps to use common steps when working with BSF and helps to get the connection to OpenOffice. The "::requires UNO.CLS" line at the end of the ooRexx script makes this support available.

```
36 /* ############################################################ */
37
38 /* open the "Database file" */
39 clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
40 openOds=clz~loadDocument("Kunden.ods")
41
42 /* create a userData directory which will be passed to Rexx */
43 userData=.directory~new
44
45 /* create exit possibility and create Rexx event handler */
46 rexxCloseEH=.RexxCloseAppEventHandler~new
47
48 /* create a RexxProxy for our Rexx event handler*/
49 jal="java.awt.event.ActionListener"
50 jwl="java.awt.event.WindowListener"
51 rpCloseEH=BsfCreateRexxProxy(rexxCloseEH, ,jal,jwl)
52
53 /* create a framed window and save it for later use */
54 GLayout =.bsf~new("java.awt.GridLayout",16,2)
55 pm = "Prüfungsverwaltung: Motoryachting"
56 window = .bsf~new("java.awt.Frame",pm)
57 window~addWindowListener(rpCloseEH)
58 userData~window=window
59
60 /* create labels */
```

```
 61 label1  = .bsf~new("java.awt.Label",'Titel')
 62 label2  = .bsf~new("java.awt.Label",'Vorname')
 63 label3  = .bsf~new("java.awt.Label",'Nachname')
 64 label4  = .bsf~new("java.awt.Label",'Strasse')
 65 label5  = .bsf~new("java.awt.Label",'PLZ')
 66 label6  = .bsf~new("java.awt.Label",'Ort')
 67 label7  = .bsf~new("java.awt.Label",'Land')
 68 label8  = .bsf~new("java.awt.Label",'Geburtstag')
 69 label9  = .bsf~new("java.awt.Label",'10m')
 70 label10 = .bsf~new("java.awt.Label",'10m SF')
 71 label11 = .bsf~new("java.awt.Label",'International')
 72 label12 = .bsf~new("java.awt.Label",'Telefon')
 73 label13 = .bsf~new("java.awt.Label",'email')
 74 label20 = .bsf~new("java.awt.Label",' ')
 75
 76 /* create text fields */
 77 titel     = .bsf~new("java.awt.TextField")
 78 firstname = .bsf~new("java.awt.TextField")
 79 surname   = .bsf~new("java.awt.TextField")
 80 street    = .bsf~new("java.awt.TextField")
 81 postcode  = .bsf~new("java.awt.TextField")
 82 town      = .bsf~new("java.awt.TextField")
 83 country   = .bsf~new("java.awt.TextField")
 84 birthday  = .bsf~new("java.awt.TextField")
 85 tenm      = .bsf~new("java.awt.TextField")
 86 tenmsf    = .bsf~new("java.awt.TextField")
 87 intern    = .bsf~new("java.awt.TextField")
 88 tel       = .bsf~new("java.awt.TextField")
 89 email     = .bsf~new("java.awt.TextField")
 90
 91 /* create userData for use with the store routine */
 92 userData~titel      = titel
 93 userData~firstname  = firstname
 94 userData~surname    = surname
 95 userData~street     = street
 96 userData~postcode   = postcode
 97 userData~town       = town
 98 userData~country    = country
 99 userData~birthday   = birthday
100 userData~tenm       = tenm
101 userData~tenmsf     = tenmsf
102 userData~intern     = intern
103 userData~tel        = tel
```

```
104 userData~email      = email
105
106 /* create buttons */
107 save   = .bsf~new("java.awt.Button",'Speichern')
108 app    = .bsf~new("java.awt.Button",'Anträge erstellen')
109 pruef  = .bsf~new("java.awt.Button",'Prüfungsliste')
110 emailb = .bsf~new("java.awt.Button",'e-mail')
111 exit   = .bsf~new("java.awt.Button",'Exit')
112
113 /* create action listener for the "Exit" button */
114 exit~addActionListener(rpCloseEH)
115
116 /* create, add the action listener for the "Speichern" button */
117 rp=BsfCreateRexxProxy(.RexxSpeichernEventHandler~new,UserData,jal)
118 save~addActionListener(rp)
119
120 /* create, add the action listener for
121 the "Anträge erstellen" button */
122 an=BsfCreateRexxProxy(.RexxApplicationEventHandler~new,UserData,jal)
123 app~addActionListener(an)
124
125 /* create, add the action listener for the "Prüfungsliste" button */
126 pr=BsfCreateRexxProxy(.RexxPruefungEventHandler~new,UserData,jal)
127 pruef~addActionListener(pr)
128
129 /* create, add the action listener for the "e-mail" button */
130 ml=BsfCreateRexxProxy(.RexxMailEventHandler~new,UserData,jal)
131 emailb~addActionListener(ml)
132
133 /* add button to framed window, pack it */
134 window ~setLayout(GLayout)
135 window ~~add(label1) ~~add(titel)
136 window ~~add(label2) ~~add(firstname)
137 window ~~add(label3) ~~add(surname)
138 window ~~add(label4) ~~add(street)
139 window ~~add(label5) ~~add(postcode)
140 window ~~add(label6) ~~add(town)
141 window ~~add(label7) ~~add(country)
142 window ~~add(label8) ~~add(birthday)
143 window ~~add(label9) ~~add(tenm)
144 window ~~add(label10)~~add(tenmsf)
145 window ~~add(label11)~~add(intern)
146 window ~~add(label12)~~add(tel)
```

```
147 window ~~add(label13)~~add(email)
148 window ~~add(save)    ~~add(pruef)
149 window ~~add(app)     ~~add(emailb)
150 window ~~add(label20)~~add(exit)
151 window ~~pack~~setVisible(.true)~~toFront
152
153 /* set initial size, location, make visible and put it to front */
154 window ~~setSize(500,500) ~~setLocation(350,50)
155 window ~~setVisible(.true) ~~toFront
156
157 /* wait until we are allowed to end the program */
158 rexxCloseEH~waitForExit
159
160 ::requires UNO.CLS
```

Figure 14: Main part

5.3 Store customers

A very important step of the whole process is to store the candidates. The “speichern” event handler (line 194 - 197) waits till the “Speichern” button is pressed by the user, to trigger the “routine speichern”.

The routine uses the “userData” array to store all entries of the main mask. All data from the different fields are now stored into the variables kc1 – kc13 (line 204 – 216).

To avoid that empty lines are stored into the “Kunden.ods” document, line 219 checks if the main fields are empty. If yes an error message is shown and the program returs to the main part to wait for further user actions.

To get the first free row of the “Kunden.ods” file, the routine “lastrow” is called in line 228. The result is stored to the variable “x”.

The next step is to import the appropriate class of the ODFToolkit. All classes, sub classes and methods are listed in the ODFDOM API documentation in detail [OdAp02]. The method `"loadDocument"` is passed on to the variable `"openOds"` and thus opens

the document which is defined in the argument of the method, in this case the spreadsheet document "Kunden.ods" which acts as a database. The first sheet is called in line 234.

Now all the stored information about the candidates has to be written into the cells of the next free row. This is triggered by the "x" variable.

After writing the information the document has to be saved. The method "save" is doing this in line 265.

To make it clear for the user what has happened a message box is shown. A Java method is used for this task. The method "messageBox" can have three arguments. The first is the Information written in the box. The second is the label, and the last one is the type of the box which is displayed by an icon, in this example an information box (see following figure). The second and third argument is voluntary. There are five possible types [JaMb01]:

- ERROR_MESSAGE
- INFORMATION_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE
- PLAIN_MESSAGE (without an argument)

Figure 15: Infobox

The "return" instruction finishes the routine.

```
192 /* ############################################################## */
193 /* Rexx event handler "speichern" */
```

```
194 ::class RexxSpeichernEventHandler
195 ::method actionPerformed
196   use arg eventObject, slotDir
197   call speichern slotDir~userData
198
199 /* ############################################################## */
200 ::routine speichern
201    use arg userData
202
203 /* read data and store it */
204 kc1 = userData~titel~getText
205 kc2 = userData~firstname~getText
206 kc3 = userData~surname~getText
207 kc4 = userData~street~getText
208 kc5 = userData~postcode~getText
209 kc6 = userData~town~getText
210 kc7 = userData~country~getText
211 kc8 = userData~birthday~getText
212 kc9 = userData~tenm~getText
213 kc10= userData~tenmsf~getText
214 kc11= userData~intern~getText
215 kc12= userData~tel~getText
216 kc13= userData~email~getText
217
218 /* check if fields are filled with data */
219 if kc2=""|kc3=""|kc4=""|kc5=""|kc6="" then
220 do
221  /* show error message */
222  .bsf.dialog~messageBox("Alle Felder ausfüllen!", ,"warning")
223  return
224 end
225
226 else
227
228 call lastrow
229 x=result
230
231 clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
232 openOds=clz~loadDocument("Kunden.ods")
233
234 odfTable=openOds~getTableList~get(0)
235
236 /* write information to the document */
```

```
237   odfcell=odfTable~getCellByPosition(0,x)
238   odfcell~setStringValue(kc1)
239   odfcell=odfTable~getCellByPosition(1,x)
240   odfcell~setStringValue(kc2)
241   odfcell=odfTable~getCellByPosition(2,x)
242   odfcell~setStringValue(kc3)
243   odfcell=odfTable~getCellByPosition(3,x)
244   odfcell~setStringValue(kc4)
245   odfcell=odfTable~getCellByPosition(4,x)
246   odfcell~setStringValue(kc5)
247   odfcell=odfTable~getCellByPosition(5,x)
248   odfcell~setStringValue(kc6)
249   odfcell=odfTable~getCellByPosition(6,x)
250   odfcell~setStringValue(kc7)
251   odfcell=odfTable~getCellByPosition(7,x)
252   odfcell~setStringValue(kc8)
253   odfcell=odfTable~getCellByPosition(8,x)
254   odfcell~setStringValue(kc9)
255   odfcell=odfTable~getCellByPosition(9,x)
256   odfcell~setStringValue(kc10)
257   odfcell=odfTable~getCellByPosition(10,x)
258   odfcell~setStringValue(kc11)
259   odfcell=odfTable~getCellByPosition(11,x)
260   odfcell~setStringValue(kc12)
261   odfcell=odfTable~getCellByPosition(12,x)
262   odfcell~setStringValue(kc13)
263
264 /* save the document */
265 openOds~save("Kunden.ods")
266
267 /* show message what happened */
268 .bsf.dialog~messageBox("Kandidat gespeichert.", ,"information")
269
270 return
```

Figure 16: Routine save candidate

5.4 Generate exam list

Another task is to generate a list with all candidates. It is needed for the registration of the exam. Not all of the information which has been stored is needed on this list. The first part of the routine “pdf” works quite similar to the “speichern” routine. The difference is that the data is retrieved from the “kunden.ods” document.

The first part is the event handler which calls the routine. The result of the row counting has to be reduced by 1 to get the number of entries and not the number of the first empty row as the routine “lastrow” delivers. We have to take in mind that the numbering of the rows starts with “0” and the first row contains the label of the column.

Then the master document for the exam list is loaded. Afterwards the “kunden.ods” document is loaded.

In line 441 – 466 a loop first reads the relevant data from “Kunden.ods” and then stores it into the master document. At the end the list is stored as ”pruefung.ods”.

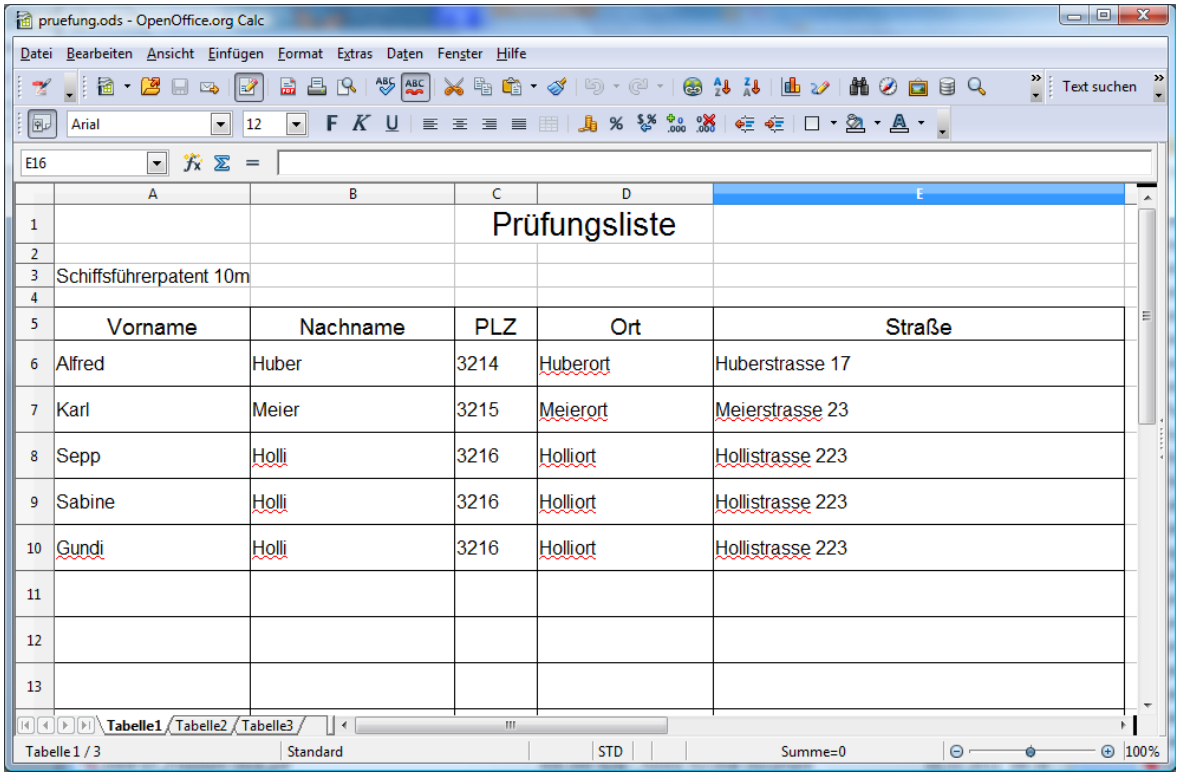


Figure 17: Example pruefung.ods

```
418 /* ############################################################## */
419 /* Rexx event handler "pruefung" */
420 ::class RexxPruefungEventHandler
421 ::method actionPerformed
422   use arg eventObject,slotDir
423   call pdf slotDir~userData
424
425 /* ############################################################## */
426 ::routine pdf
427
428 call lastrow
429 x=result-1
430
431 /* open the master document */
432 clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
433 pruefung=clz~loadDocument("pruefungsmaster.ods")
434 pruefungTable=pruefung~getTableList~get(0)
435
436 /* open the import file */
437 clzs=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
438 kunden=clzs~loadDocument("kunden.ods")
439 kundenTable=kunden~getTableList~get(0)
440
441 do i=1 to x
442   /* read necessary data */
443   odfcell=kundenTable~getCellByPosition(1,i)
444   firstname=odfcell~getStringValue()
445   odfcell=kundenTable~getCellByPosition(2,i)
446   surname=odfcell~getStringValue()
447   odfcell=kundenTable~getCellByPosition(3,i)
448   street=odfcell~getStringValue()
449   odfcell=kundenTable~getCellByPosition(4,i)
450   postcode=odfcell~getStringValue()
451   odfcell=kundenTable~getCellByPosition(5,i)
452   town=odfcell~getStringValue()
453
454   /* write new information */
455   y=i+4
456   odfcell=pruefungTable~getCellByPosition(0,y)
457   odfcell~setStringValue(firstname)
458   odfcell=pruefungTable~getCellByPosition(1,y)
459   odfcell~setStringValue(surname)
460   odfcell=pruefungTable~getCellByPosition(2,y)
```

```
461    odfcell~setStringValue(postcode)
462    odfcell=pruefungTable~getCellByPosition(3,y)
463    odfcell~setStringValue(town)
464    odfcell=pruefungTable~getCellByPosition(4,y)
465    odfcell~setStringValue(street)
466 end
467
468 /* save the outputfile */
469 pruefung~save("pruefung.ods")
```

Figure 18: Routine pdf / part create Exam list

5.5 Convert to PDF

The second part of the routine “pdf” takes the previously created spreadsheet document and converts it into a PDF document.

Prüfungsliste

Schiffsführerpatent 10m

Vorname	Nachname	PLZ	Ort	Straße
Alfred	Huber	3214	Huberort	Huberstrasse 17
Karl	Meier	3215	Meierort	Meierstrasse 23
Sepp	Holli	3216	Holliort	Hollistrasse 223
Sabine	Holli	3216	Holliort	Hollistrasse 223
Gundi	Holli	3216	Holliort	Hollistrasse 223

Figure 19: Example PDF exam list

The second part of the “pdf” routine starts with the instruction to wait for two seconds. A short look at the code, which is executed prior to that, helps to understand the necessity of this instruction. In line 469 the generated ODF document is saved. To save and close the document takes a little bit of time. If the program does not wait a little bit, the opening instruction of the document in line 475 may cause an error on some systems, because the document is still in use.

The Apache OpenOffice API is included in the Aoo installation and offers a lot of methods to handle Aoo documents [ApAp01].

First a connection to the Aoo server is established. The document is opened invisible in the background. The conversion to PDF is realized by using a PDF export filter in the method “storeToURL”. Afterwards the document is closed and the message box reports the result for the user.

```
471 call sysSleep 2
472
473 /* initialize connection to the AOo server, get its XContext */
474 xContext = UNO.connect()
475 url=uno.convertToUrl(directory()"/pruefung.ods")
476
477 /* load the calc file hiddenly */
478 xDesktop = UNO.createDesktop(xContext)
479 xComponentLoader = xDesktop~XComponentLoader
480 props = bsf.createJavaArray(.UNO~propertyValue, 1)
481 props[1]=uno.createProperty("Hidden",box("boolean", .true))
482 calcComponent = xComponentLoader ,
483 ~loadComponentFromURL(url,"_blank",0,props)
484
485 /* convert file to PDF using a PDF export filter
486  to be applied in "storeToURL" and get xStorable interface
487  create Java array for two PropertyValue entries*/
488 xStorable = calcComponent~XStorable
489 props = bsf.createJavaArray(.UNO~propertyValue, 2)
490 props[1]=uno.createProperty("FilterName","writer_pdf_Export")
491 props[2]=uno.createProperty("CompressMode",2)
492
493 /* create output file name and store it with props to URL */
494 storeURL = substr(url, 1, lastpos(".",url)) || "pdf"
```

```
495 xStorable~storeToUrl(storeURL,props)
496
497 /* close calcComponent */
498 x_Closeable = calcComponent~XCloseable
499 x_Closeable~close(.true)
500
501 /* show message what happened */
502 .bsf.dialog~messageBox("Liste gespeichert!", ,"information")
503
504 return
```

Figure 20: Routine pdf / part create pdf

5.6 Generate exam form

The “createapplication” routine starts similar to the first part of the “pdf” routine. The “Kunden.ods” document is opened with the “loadDocument” method from the ODFDOM class. From line 303 till 411 a loop handles the data. For every candidate an extra document will be created.

There are different parts in this loop. The first part retrieves the data from the “Kunden.ods” document. Otherwise the program would cause an error because of the non existing files. The first parameter of the “stream” instruction defines the filename. The second defines the type of the action. “C” means command which is specified in parameter three. One of the possible parameters.is “query exists”. If the defined file exists it returns the full path specification of the named stream, else a null string [OoRe02].

The second part from line 342 – 357 checks if there are the adequat files for the picture and the signature. If not a messagebox with an error message is shown and the routine is finished.

Starting with line 360 the third part starts. In contrast to the first part a different Java class is used to open the document. The method “openDocument” is from the Simple API class. In the next lines the data is stored into the appropriate cells of the master document. The reason why a different Java class is used shows the next part of the loop.

The “setImage” method saves an image directly to a cell of the spreadsheet document. This is not possible with the methods of the ODFDOM class, but the Simple API class offers this possibility.

When an image has to be inserted, the correct path to the desired image has to be specified. Therefore a URI is used. In Java a URI[11] object is described by the class “java.net.URI” [JaUr01]. As defined in the class description there are different types of URI specifications. In this example a relative path is used. This means that the path to the desired document describes just the difference to the path from which it was called. Since the image in this example is in the same folder, only the file name is needed [Maye01, page 18].

The filename has to follow the following syntax: “p”firstname surname.jpg.

According to this syntax the pictures of the candidates have to be stored in the same directory as the main program in advance.

Till line 407 further data is stored into some cells and also another image, for the signature of the candidates.

At the end, the document has to be saved and a message box comments the result.

The following figure shows an example of the created document.

11 Uniform Resource Identifier

An den Landeshauptmann von Niederösterreich als Schifffahrtsbehörde

ANTRAG AUF ZULASSUNG ZUR PRÜFUNG FÜR

	SCHIFFSFÜHRERPATENT 20m SEEN UND FLÜSSE
x	SCHIFFSFÜHRERPATENT 10m
	SCHIFFSFÜHRERPATENT 10m SEEN UND FLÜSSE
	SCHIFFSFÜHRERPATENT RAFT

ANTRAG AUF AUSSTELLUNG

x	INTERNATIONALES ZERTIFIKAT FÜR FÜHRER VON SPORTFAHRZEUGEN

ANTRAGSTELLER

Akademischer Grad	
Familienname	Huber
Vorname	Alfred
Wohnadresse	3214 Huberort Huberstrasse 17
Geburtsort und Datum	02.05.1980
Geburtsstaat	
Staatsangehörigkeit	

Foto

ZUSTELLADRESSE

Straße	Huberstrasse 17
PLZ, Ort	3214 Huberort
Telefon	0698123546

Unterschrift

1 Nov 2012
Datum Unterschrift

Figure 21: Example generated application form

```
285 /* ############################################################## */
286 /* Rexx event handler "application" */
287 ::class RexxApplicationEventHandler
288 ::method actionPerformed
289   use arg eventObject,slotDir
290   call createapplication slotDir~userData
291
292
293 /* ############################################################## */
294 :: routine createapplication
295
296 clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
297 openOds=clz~loadDocument("Kunden.ods")
298 odfTable=openOds~getTableList~get(0)
299
300 call lastrow
301 x=result-1
302
303 do i=1 to x
304
305   /* copy content of a cell */
306   odfcell=odfTable~getCellByPosition(0,i)
307   titel=odfcell~getStringValue()
308
309   odfcell=odfTable~getCellByPosition(1,i)
310   firstname=odfcell~getStringValue()
311
312   odfcell=odfTable~getCellByPosition(2,i)
313   surname=odfcell~getStringValue()
314
315   odfcell=odfTable~getCellByPosition(3,i)
316   street=odfcell~getStringValue()
317
318   odfcell=odfTable~getCellByPosition(4,i)
319   postcode=odfcell~getStringValue()
320
321   odfcell=odfTable~getCellByPosition(5,i)
322   town=odfcell~getStringValue()
323
324   odfcell=odfTable~getCellByPosition(6,i)
325   country=odfcell~getStringValue()
326
327   odfcell=odfTable~getCellByPosition(7,i)
```

```
328   birthday=odfcell~getStringValue()
329
330   odfcell=odfTable~getCellByPosition(8,i)
331   tenm=odfcell~getStringValue()
332
333   odfcell=odfTable~getCellByPosition(9,i)
334   tenmsf=odfcell~getStringValue()
335
336   odfcell=odfTable~getCellByPosition(10,i)
337   intern=odfcell~getStringValue()
338
339   odfcell=odfTable~getCellByPosition(11,i)
340   tel=odfcell~getStringValue()
341
342   /* check if foto or signature files are there */
343   fotoname = "p"firstname""surname".jpg"
344   if stream(fotoname,"C",query exists) = "" then
345   do
346    fi = "Datei: " fotoname " nicht gefunden!"
347    .bsf.dialog~messageBox( fi, ,"warning")
348    return
349   end
350
351   signaturname = "u"firstname""surname".jpg"
352   if stream(signaturname,"C",query exists) = "" then
353   do
354    si = "Datei: " signaturname " nicht gefunden!"
355    .bsf.dialog~messageBox( si, ,"warning")
356    return
357   end
358
359   /* open master document and fill data */
360   clzm=bsf.import("org.odftoolkit.simple.SpreadsheetDocument")
361   openAntrag=clzm~loadDocument("antragsmaster.ods")
362   antragTable=openAntrag~getTableList~get(0)
363
364   odfcell=antragTable~getCellByPosition(0,5)
365   odfcell~setStringValue(tenm)
366
367   odfcell=antragTable~getCellByPosition(0,6)
368   odfcell~setStringValue(tenmsf)
369
370   odfcell=antragTable~getCellByPosition(0,11)
```

```
371   odfcell~setStringValue(intern)
372
373   odfcell=antragTable~getCellByPosition(2,14)
374   odfcell~setStringValue(titel)
375
376   odfcell=antragTable~getCellByPosition(2,15)
377   odfcell~setStringValue(surname)
378
379   odfcell=antragTable~getCellByPosition(2,16)
380   odfcell~setStringValue(firstname)
381
382   odfcell=antragTable~getCellByPosition(2,17)
383   odfcell~setStringValue(postcode town street)
384
385   odfcell=antragTable~getCellByPosition(2,18)
386   odfcell~setStringValue(birthday)
387
388   /* add a picture */
389   odfcell=antragTable~getCellByPosition(3,22)
390   odfcell ,
391   ~setImage(.bsf~new("java.net.URI",fotoname))
392
393   odfcell=antragTable~getCellByPosition(2,29)
394   odfcell~setStringValue(street)
395
396   odfcell=antragTable~getCellByPosition(2,30)
397   odfcell~setStringValue(postcode town)
398
399   odfcell=antragTable~getCellByPosition(2,31)
400   odfcell~setStringValue(tel)
401
402   odfcell=antragTable~getCellByPosition(1,33)
403   odfcell~setStringValue(Date())
404
405   odfcell=antragTable~getCellByPosition(2,33)
406   odfcell ,
407   ~setImage(.bsf~new("java.net.URI",signaturname))
408
409   /* save the document */
410   openAntrag~save("antrag "firstname surname".ods")
411 end
412
413 /* show message what happened */
```

```
414 .bsf.dialog~messageBox(x" Anträge gespeichert!", ,"information")
415
416 return
```

Figure 22: Routine createapplication

5.7 Send email

The routine “mail” takes the exam list (“pruefung.pdf”) from chapter 5.5 and sends it via email. The receiver address is specified in line 517.

JavaMail is a MUA, as explained in chapter 2.9, and needs a MTA to fulfill its tasks.

Therefore a dummy mail account has been created to test the functionality. It's a gmail account.

- Address: `mailrexx@gmail.com`
- Password: `mailrexx123`

The account properties from line 520 – 529 are the relevant ones for a gmail account. The outgoing mail server (SMTP[12]) requires following configuration [GoMa01].

- Host: smtp.gmail.com
- Use Authentication: Yes
- Port for TLS/STARTTLS: 587

The configuration has to be adapted for other email accounts.

Line 533 imports the appropriate class to get the e-mail support. The subject of the e-mail is defined in line 538. In line 540 it is possible to add some information which should be sent to the receiver. This is the body of the e-mail.

The next part defines the relevant data for the attachment.

12 Simple Mail Transfer Protocol

Line 551 – 555 combines the text and the attachment and creates the multipart message [MiMe01].

The receiver is added to the message in line 567. At the end the message is sent with all the defined options.

The message box comments the action which has been taken for the user. Figure 23 shows an example how the sent e-mail looks in a gmail account. All elements which have been defined are there. The subject “Test”, the bodytext ”Kandidatenliste” and the attachment “pruefung.pdf”. And the sender of the e-mail is the dummy account “mailrexx@gmail.com”.

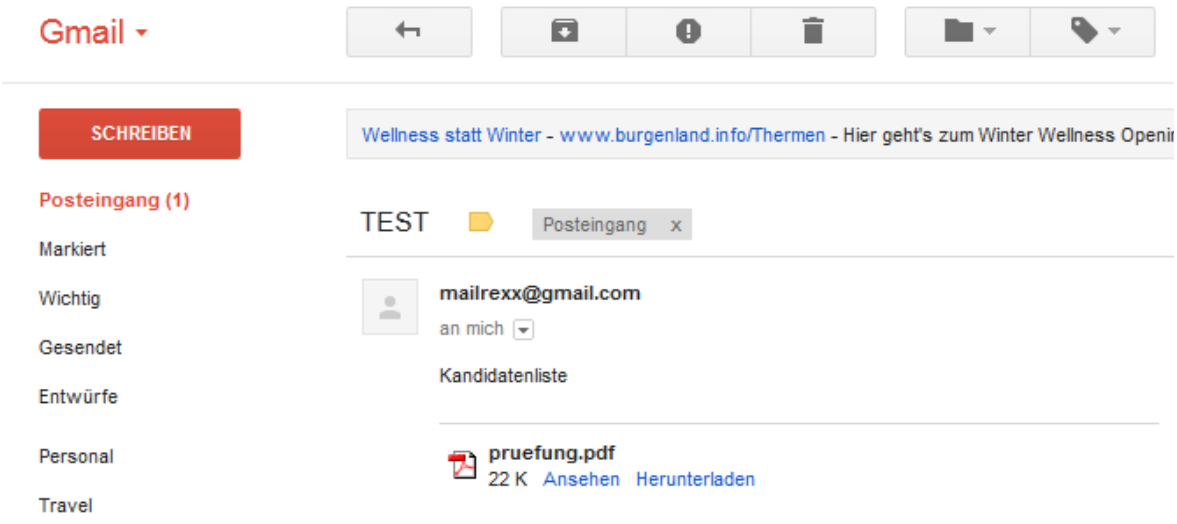


Figure 23: Example mail in a gmail account

```
506 /* ############################################################## */
507 /* Rexx event handler "mail"  */
508 ::class RexxMailEventHandler
509 ::method actionPerformed
510   use arg eventObject,slotDir
511   call mail slotDir~userData
512
513 /* ############################################################## */
514 ::routine mail
515
```

```
516 /* define receiver */
517 empfaenger = "guentermayer04@gmail.com"
518
519 /* define sendermailaddress */
520 sendermail = "mailrexx@gmail.com"
521
522 /* set sender mail account properties */
523 mail=.BSF~new("java.util.Properties")
524 mail~setProperty("mail.transport.protocol","smtps")
525 mail~setProperty("mail.smtp.port","587")
526 mail~setProperty("mail.smtp.host","smtp.gmail.com")
527 mail~setProperty("mail.smtp.user",sendermail)
528 mail~setProperty("mail.smtp.password","mailrexx123")
529 mail~setProperty("mail.smtp.auth","true")
530
531 /* The Session class collects together properties
532  and defaults used by the mail API's.*/
533 impsess=BSF.import("javax.mail.Session")
534 session=impsess~getDefaultInstance(mail)
535
536 /* MimeMessage: */
537 msge=.BSF~new('javax.mail.internet.MimeMessage',session)
538 msge~setSubject("TEST")
539 testtext=.BSF~new('javax.mail.internet.MimeBodyPart')
540 msgbodypart=testtext~setText("Kandidatenliste")
541
542 /* Attachment */
543 filename = uno.convertFromUrl(directory()"/pruefung.pdf")
544 testatt=.BSF~new('javax.mail.internet.MimeBodyPart')
545 source=.BSF~new('javax.activation.FileDataSource',filename)
546 imphand=.BSF~new('javax.activation.DataHandler',source)
547 testatt~setDataHandler(imphand)
548 testatt~setFileName("pruefung.pdf")
549
550 /* create Mulitpart */
551 multipart=.BSf~new('javax.mail.internet.MimeMultipart')
552 multipart~addBodyPart(testtext)
553 multipart~addBodypart(testatt)
554 fileDataSource=.bsf~new('javax.activation.FileDataSource',filename)
555 narf=fileDataSource~getContentType
556
557 /* --> error message: no content!!! */
558 msge~setContent(multipart,"application/octet-stream")
```

```
559
560 /* InternetAdresse of receiver */
561 new= bsf.importClass("javax.mail.internet.MimeMessage$RecipientType")
562 ia=.bsf~new("javax.mail.internet.InternetAddress",empfaenger)
563
564 /* define receiver */
565 msge~addRecipient(new~to,ia)
566 sender = .bsf~new("javax.mail.internet.InternetAddress",sendermail)
567 msge~setFrom(sender)
568
569 /* Get a Transport object that implements
570 this user's desired Transport protcol */
571 transport =session~getTransport("smtps")
572 transport~connect("smtp.gmail.com",465,sendermail,"mailrexx123");
573 transport~sendMessage(msge,msge~getRecipients(new~to))
574
575 /* show message what happened */
576 .bsf.dialog~messageBox("e-mail wurde versendet!",,"information")
577
578 return
```

Figure 24: Routine mail

5.8 Count Database entries

Every time when there is a connection to the “Kunden.ods” file, it is necessary to know how much entries are in the list. The task of the routine “lastrow” is to count the number of entries.

It starts in line 275 with opening the “Kunden.ods” document and focusing at the first sheet, where the candidates are stored. In line 280 the second column is selected. It is the column where the firstnames are stored. The first column was not chosen because not every person owns a title. The result of the counting is stored in the variable “x” which will be returned, to the originator of the routine, in line 283. The routine is called from following parts of the program:

- routine “speichern”
- routine ”application”
- routine “pdf”

```
272 /* ################################################################ */
273 :: routine lastrow
274
275 clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
276 openOds=clz~loadDocument("Kunden.ods")
277 odfTable=openOds~getTableList~get(0)
278
279 /* count the number of entries */
280 odfcollumn=odfTable~getColumnList()~get(1)
281 x=odfcollumn~getCellCount()
282
283 return x
```

Figure 25: Routine lastrow

5.9 Prerequisites

When running the program some essential points have to be observed. The ooRexx file has the file extension `.rxj`, which means it needs Java support.

The main program uses the files `"antragsmaster.ods"`, `"pruefungsmaster.ods"` and `“Kunden.ods”` so the three files have to be located in the same directory.

These four files are essential to run the program:

- pruefungsverwaltung.rxj (the main program)
- antragsmaster.ods
- pruefungsmaster.ods

- Kunden.ods

  _____

- The pictures of the candidates also have to be stored in the same directory before the applications are created. The syntax of the filename has to be as follows: p (for picture) firstname surname.jpg (without spaces)
  Example: Mr.Alex Test --> palextest.jpg

  _____

- The syntax of the signature files works similar as for the pictures.
  u firstname surname.jpg (without spaces)
  Example: Mr.Alex Test --> ualextest.jpg

  _____

- All generated files are stored in the same directory.

  _____

- In Line 517 the email address of the receiver has to be stored.
- Line 520 - 529 contains the email data of the sender.

  _____

- Pressing the button "Speichern" writes a new candidate into the file "Kunden.ods".
- "Prüfungsliste" generates the files "pruefung.ods" and "pruefung.pdf".
- "Anträge erstellen" produces for each candidate a file that looks like "antrag firstname surname.ods".

6 Conclusion

The ODF toolkit includes very powerful tools to handle ODF documents.

The documentation is not very comprehensive and so it was not easy to familiarize with the different methods. Also the combination of other Java classes like JavaMail was a bich challenge but very interesting. It was a great challenge to combine these different methods and tools to create a result that is not only a theoretical construct but rather a real useful tool.

The result is realy useful. Most parts of the program have been tested in a real situation in September 2012. For the exam, at the nautical authority in Lower Austria, have been twelve candidates registered. Previouesly it took about two hours to register this number of candidates. The program helped to make the work of creating the application forms easier and it tooks about half an hour.

The fact, that the ODF toolkit has been welcomed in the incubator by the Apache Software Foundation, allocates resources and opportunities, to develop the tools and the corresponding documentation. This may enable future developers to use these great tools more often.

Hopefully, this work contributes a small piece to this.

7 List of references

[AdPd01] Adobe Systems
http://www.adobe.com/
[Accessed 18 October 2012]

[AdPd02] Portable Document Format
http://en.wikipedia.org/wiki/Portable_Document_Format
[Accessed 18 October 2012]

[ApAp01] API Project Structure
http://www.openoffice.org/api/modules.html
[Accessed 28 October 2012]

[ApAp02] Apache OpenOffice API module star
http://www.openoffice.org/api/docs/common/ref/com/sun/star/module-ix.html
[Accessed 28 October 2012]

[[ApSo01] Apache Software Foundation
http://www.apache.org
[Accessed 15 August 2012]

[ApSo02] Apache Software Top Level
http://www.openoffice.org/de/presse/pressemitteilung_20121018_Graduierung.html
[Accessed 18 October 2012]

[ApXe01] Xerces2 Java Parser
http://xerces.apache.org/xerces2-j/
[Accessed 15 September 2012]

[BsOr01] BSF4ooRexx
http://bsf4oorexx.sourceforge.net/
[Accessed 15 August 2012]

[Flat01, p. 4] Flatscher, Rony G.: Camouflaging Java as Object REXX
International Rexx Symposium, Böblingen, 05 2004

[GoMa01] Google mail, Set up POP in mail clients
http://support.google.com/mail/bin/answer.py?hl=en&answer=13287
[Accessed 28 September 2012]

[HeWo01] Hello World Rexx
http://www.rexxla.org/rexxlang/rexxtut.html
[Accessed 05 August 2012]

[HeWo02] Hello World Java
http://introcs.cs.princeton.edu/java/11hello/HelloWorld.java.html
[Accessed 05 August 2012]

[JaEe01] Java Platform, Enterprise Edition
http://en.wikipedia.org/wiki/Java_EE
[Accessed 17 September 2012]

[JaMa01] JavaMail
http://en.wikipedia.org/wiki/JavaMail
[Accessed 17 September 2012]

[JaMb01] Java Message Box
http://java.about.com/od/UsingDialogBoxes/a/Creating-A-Message-Dialog-Box-Part-I.htm
[Accessed 19 September 2012]

[JaSe01] Java Platform, Standard Edition
http://en.wikipedia.org/wiki/Java_SE
[Accessed 17 September 2012]

[JaUr01] java.net.URI,
http://download.oracle.com/javase/1.4.2/docs/api/java/net/URI.html,
[Accessed 29 October 2012]

[JaVa01] Wikipedia: Java (programming language), 2012
http://en.wikipedia.org/wiki/Java_%28programming_language%29
[Accessed 31 August 2012]

[LgPl01] GNU LESSER GENERAL PUBLIC LICENSE
http://www.gnu.org/licenses/lgpl.html
[Accessed 02 June 2012]

[LiOf01] LibreOffice
http://www.libreoffice.org/
[Accessed 17 August 2012]

[Maye01] Mayer Günter: Scripting the ODF Toolkit (ODFDOM), 2011
http://wi.wu.ac.at:8002/rgf/diplomarbeiten/
[Accessed 10 October 2012]

[MiMe01] MIME
http://en.wikipedia.org/wiki/MIME
[Accessed 22 October 2012]

[NoSb01] Niederösterreichische Landesregierung / Schiffsführung
http://www.noe.gv.at/Verkehr-Technik/Schifffahrt/Schiffsfuehrung/schifffahrt_patent.html
[Accessed 02 August 2012]

[OaOp01] Oasis (Organization for the Advancement of Structured Information Standards)
https://www.oasis-open.org/
[Accessed 12 August 2012]

[OdAp01] Approval OpenDocument Format
https://www.oasis-open.org/news/pr/odf-1-2-approval
[Accessed 17 October 2012]

[OdAp02] ODFDOM Javadoc
http://incubator.apache.org/odftoolkit/odfdom/index.html
[Accessed 17 October 2012]

[OdMi01] MIME Types and Extensions for OpenDocument Documents Table1.1
http://books.evc-cit.info/odbook/ch01.html
[Accessed 27 October 2012]

[OoPr01] Object-oriented programming
http://en.wikipedia.org/wiki/Object-oriented_programming
[Accessed 30 June 2012]

[OoRe01] Open Object Rexx
http://www.oorexx.org/about.html
[Accessed 25 June 2012]

[OoRe02] Open Object Rexx Referenz
http://www.oorexx.org/docs/rexxref/x23579.htm
[Accessed 27 October 2012]

[PrPa01] Programming paradim
http://en.wikipedia.org/wiki/Programming_paradigm
[Accessed 10 October 2012]

[ReXx01] REXX, 2012
http://en.wikipedia.org/w/index.php?title=REXX&oldid=471379104
[Accessed 20 April 2012]

[SiAp01] Simple API (Simple Java API for ODF)
http://incubator.apache.org/odftoolkit/simple/index.html
[Accessed 1 November 2012]

[ViEd01] Vim the editor
http://www.vim.org/download.php
[Accessed 4 November 2012]

[WiSe01] Windows 7
http://windows.microsoft.com/de-AT/windows/downloads/windows-7
[Accessed 05 June 2012]

[WiVi01] Windows Vista
http://windows.microsoft.com/de-AT/windows/downloads/windows-vista
[Accessed 05 June 2012]

8 Download Links

Acrobat Reader Adobe Acrobat Reader
http://get.adobe.com/de/reader/
[Accessed 22 July 2012]

AOo Apache OpenOffice
http://www.openoffice.org/download/
[Accessed 12 July 2012]

BSF4ooRexx Bean Scripting Framework for ooRexx
http://sourceforge.net/projects/bsf4oorexx/
[Accessed 14 Octobre 2012]

J2SE **S**un Java2 Standard Edition
http://www.java.com
[Accessed 19 September 2012]

JavaMail JavaMail
http://www.oracle.com/technetwork/java/javamail/index.html
[Accessed 12 September 2012]

ODF Java odftoolkit 0.5
http://incubator.apache.org/odftoolkit/downloads.html
[Accessed 12 September 2012]

OORexx OpenObjectRexx
http://sourceforge.net/projects/oorexx/
[Accessed 12 September 2012]

Simple API Simple API included in Java odftoolkit 0.5
http://incubator.apache.org/odftoolkit/downloads.html
[Accessed 30 October 2012]

Xerces Java XSLT processor
http://tweedo.com/mirror/apache//xerces/j/Xerces-J-src.2.11.0-xml-schema-1.1-beta.zip
[Accessed 12 July 2012]

9 Statement

I hereby declare that I have made this work independently and without use of other than the specified resources. All parts that were taken literally or in spirit from published or unpublished writings are identified as such. The work has not been presented as part of another test in the same or similar form or extracts of it.

Baden, 2012-11-05

Ing. Günter Mayer

10 Appendix 1 Program Code

```
/* Prüfungsverwaltung created by Günter Mayer 10.11.2012 */
/*
When running the program some essential points have to be observed.
The ooRexx files have the file extension .rxj,
which means they need Java support.
The program uses the files "antragsmaster.ods",
"pruefungsmaster.ods" and "Kunden.ods"
so the three files have to be located in the same directory.
This four files are essential to run the program:
pruefungsverwaltung.rxj
antragsmaster.ods
pruefungsmaster.ods
Kunden.ods

########################
The pictures of the candidates also have to be stored
in the same directory before the applications are created.
The syntax of the filename has to be as follows:
p (for picture) firstname surname.jpg (without spaces)
Example:  Mr.Alex Test -->  palextest.jpg

########################
The syntax of the signature files works similar
as for the pictures.
u firstname surname.jpg (without spaces)
Example: Mr.Alex Test -->  ualextest.jpg

########################
All generated files are stored in the same directory.

########################
In Line 517 the email address of the receiver has to be stored.
Line 520 - 529 contains the email data of the sender.
*/
```

```
/* ############################################################## */

/* open the "Database file" */
clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("Kunden.ods")

/* create a userData directory which will be passed to Rexx */
userData=.directory~new

/* create exit possibility and create Rexx event handler */
rexxCloseEH=.RexxCloseAppEventHandler~new

/* create a RexxProxy for our Rexx event handler*/
jal="java.awt.event.ActionListener"
jwl="java.awt.event.WindowListener"
rpCloseEH=BsfCreateRexxProxy(rexxCloseEH, ,jal,jwl)

/* create a framed window and save it for later use */
GLayout =.bsf~new("java.awt.GridLayout",16,2)
pm = "Prüfungsverwaltung: Motoryachting"
window = .bsf~new("java.awt.Frame",pm)
window~addWindowListener(rpCloseEH)
userData~window=window

/* create labels */
label1  = .bsf~new("java.awt.Label",'Titel')
label2  = .bsf~new("java.awt.Label",'Vorname')
label3  = .bsf~new("java.awt.Label",'Nachname')
label4  = .bsf~new("java.awt.Label",'Strasse')
label5  = .bsf~new("java.awt.Label",'PLZ')
label6  = .bsf~new("java.awt.Label",'Ort')
label7  = .bsf~new("java.awt.Label",'Land')
label8  = .bsf~new("java.awt.Label",'Geburtstag')
label9  = .bsf~new("java.awt.Label",'10m')
label10 = .bsf~new("java.awt.Label",'10m SF')
label11 = .bsf~new("java.awt.Label",'International')
label12 = .bsf~new("java.awt.Label",'Telefon')
label13 = .bsf~new("java.awt.Label",'email')
label20 = .bsf~new("java.awt.Label",' ')

/* create text fields */
titel     = .bsf~new("java.awt.TextField")
firstname = .bsf~new("java.awt.TextField")
```

```
surname   = .bsf~new("java.awt.TextField")
street    = .bsf~new("java.awt.TextField")
postcode  = .bsf~new("java.awt.TextField")
town      = .bsf~new("java.awt.TextField")
country   = .bsf~new("java.awt.TextField")
birthday  = .bsf~new("java.awt.TextField")
tenm      = .bsf~new("java.awt.TextField")
tenmsf    = .bsf~new("java.awt.TextField")
intern    = .bsf~new("java.awt.TextField")
tel       = .bsf~new("java.awt.TextField")
email     = .bsf~new("java.awt.TextField")

/* create userData for use with the store routine */
userData~titel      = titel
userData~firstname  = firstname
userData~surname    = surname
userData~street     = street
userData~postcode   = postcode
userData~town       = town
userData~country    = country
userData~birthday   = birthday
userData~tenm       = tenm
userData~tenmsf     = tenmsf
userData~intern     = intern
userData~tel        = tel
userData~email      = email

/* create buttons */
save   = .bsf~new("java.awt.Button",'Speichern')
app    = .bsf~new("java.awt.Button",'Anträge erstellen')
pruef  = .bsf~new("java.awt.Button",'Prüfungsliste')
emailb = .bsf~new("java.awt.Button",'e-mail')
exit   = .bsf~new("java.awt.Button",'Exit')

/* create action listener for the "Exit" button */
exit~addActionListener(rpCloseEH)

/* create, add the action listener for the "Speichern" button */
rp=BsfCreateRexxProxy(.RexxSpeichernEventHandler~new,UserData,jal)
save~addActionListener(rp)

/* create, add the action listener for
the "Anträge erstellen" button */
```

```
an=BsfCreateRexxProxy(.RexxApplicationEventHandler~new,UserData,jal)
app~addActionListener(an)

/* create, add the action listener for the "Prüfungsliste" button */
pr=BsfCreateRexxProxy(.RexxPruefungEventHandler~new,UserData,jal)
pruef~addActionListener(pr)

/* create, add the action listener for the "e-mail" button */
ml=BsfCreateRexxProxy(.RexxMailEventHandler~new,UserData,jal)
emailb~addActionListener(ml)

/* add button to framed window, pack it */
window ~setLayout(GLayout)
window ~~add(label1) ~~add(titel)
window ~~add(label2) ~~add(firstname)
window ~~add(label3) ~~add(surname)
window ~~add(label4) ~~add(street)
window ~~add(label5) ~~add(postcode)
window ~~add(label6) ~~add(town)
window ~~add(label7) ~~add(country)
window ~~add(label8) ~~add(birthday)
window ~~add(label9) ~~add(tenm)
window ~~add(label10)~~add(tenmsf)
window ~~add(label11)~~add(intern)
window ~~add(label12)~~add(tel)
window ~~add(label13)~~add(email)
window ~~add(save)    ~~add(pruef)
window ~~add(app)     ~~add(emailb)
window ~~add(label20)~~add(exit)
window ~~pack~~setVisible(.true)~~toFront

/* set initial size, location, make visible and put it to front */
window ~~setSize(500,500) ~~setLocation(350,50)
window ~~setVisible(.true) ~~toFront

/* wait until we are allowed to end the program */
rexxCloseEH~waitForExit

::requires UNO.CLS

/* ############################################################ */
/* Rexx event handler "exit" */
::class RexxCloseAppEventHandler
```

```
/* if set to .true, then it is safe to close the app */
::method init
  expose closeApp
  closeApp = .false

/* indicates whether app should be closed */
::attribute closeApp

/* intercept unhandled events, do nothing */
::method unknown

/* event method (from ActionListener)to close app */
::method actionPerformed
  expose closeApp
  closeApp=.true

/* event method (from WindowListener) to close app */
::method windowClosing
  expose closeApp
  closeApp=.true

/* method blocks until attribute is set to .true */
::method waitForExit
  expose closeApp
  guard on when closeApp=.true

/* ################################################################ */
/* Rexx event handler "speichern" */
::class RexxSpeichernEventHandler
::method actionPerformed
  use arg eventObject, slotDir
  call speichern slotDir~userData

/* ################################################################ */
::routine speichern
   use arg userData

/* read data and store it */
kc1 = userData~titel~getText
kc2 = userData~firstname~getText
kc3 = userData~surname~getText
kc4 = userData~street~getText
```

```
kc5 = userData~postcode~getText
kc6 = userData~town~getText
kc7 = userData~country~getText
kc8 = userData~birthday~getText
kc9 = userData~tenm~getText
kc10= userData~tenmsf~getText
kc11= userData~intern~getText
kc12= userData~tel~getText
kc13= userData~email~getText

/* check if fields are filled with data */
if kc2=""|kc3=""|kc4=""|kc5=""|kc6="" then
do
 /* show error message */
 .bsf.dialog~messageBox("Alle Felder ausfüllen!", ,"warning")
 return
end

else

call lastrow
x=result

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("Kunden.ods")

odfTable=openOds~getTableList~get(0)

/* write information to the document */
  odfcell=odfTable~getCellByPosition(0,x)
  odfcell~setStringValue(kc1)
  odfcell=odfTable~getCellByPosition(1,x)
  odfcell~setStringValue(kc2)
  odfcell=odfTable~getCellByPosition(2,x)
  odfcell~setStringValue(kc3)
  odfcell=odfTable~getCellByPosition(3,x)
  odfcell~setStringValue(kc4)
  odfcell=odfTable~getCellByPosition(4,x)
  odfcell~setStringValue(kc5)
  odfcell=odfTable~getCellByPosition(5,x)
  odfcell~setStringValue(kc6)
  odfcell=odfTable~getCellByPosition(6,x)
  odfcell~setStringValue(kc7)
```

```
  odfcell=odfTable~getCellByPosition(7,x)
  odfcell~setStringValue(kc8)
  odfcell=odfTable~getCellByPosition(8,x)
  odfcell~setStringValue(kc9)
  odfcell=odfTable~getCellByPosition(9,x)
  odfcell~setStringValue(kc10)
  odfcell=odfTable~getCellByPosition(10,x)
  odfcell~setStringValue(kc11)
  odfcell=odfTable~getCellByPosition(11,x)
  odfcell~setStringValue(kc12)
  odfcell=odfTable~getCellByPosition(12,x)
  odfcell~setStringValue(kc13)

/* save the document */
openOds~save("Kunden.ods")

/* show message what happened */
.bsf.dialog~messageBox("Kandidat gespeichert.", ,"information")

return

/* ################################################################ */
:: routine lastrow

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("Kunden.ods")
odfTable=openOds~getTableList~get(0)

/* count the number of entries */
odfcollumn=odfTable~getColumnList()~get(1)
x=odfcollumn~getCellCount()

return x

/* ################################################################ */
/* Rexx event handler "application" */
::class RexxApplicationEventHandler
::method actionPerformed
  use arg eventObject,slotDir
  call createapplication slotDir~userData


/* ################################################################ */
```

```
:: routine createapplication

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("Kunden.ods")
odfTable=openOds~getTableList~get(0)

call lastrow
x=result-1

do i=1 to x

  /* copy content of a cell */
  odfcell=odfTable~getCellByPosition(0,i)
  titel=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(1,i)
  firstname=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(2,i)
  surname=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(3,i)
  street=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(4,i)
  postcode=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(5,i)
  town=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(6,i)
  country=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(7,i)
  birthday=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(8,i)
  tenm=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(9,i)
  tenmsf=odfcell~getStringValue()

  odfcell=odfTable~getCellByPosition(10,i)
```

```
intern=odfcell~getStringValue()

odfcell=odfTable~getCellByPosition(11,i)
tel=odfcell~getStringValue()

/* check if foto or signature files are there */
fotoname = "p"firstname""surname".jpg"
if stream(fotoname,"C",query exists) = "" then
do
 fi = "Datei: " fotoname " nicht gefunden!"
 .bsf.dialog~messageBox( fi, ,"warning")
 return
end

signaturname = "u"firstname""surname".jpg"
if stream(signaturname,"C",query exists) = "" then
do
 si = "Datei: " signaturname " nicht gefunden!"
 .bsf.dialog~messageBox( si, ,"warning")
 return
end

/* open master document and fill data */
clzm=bsf.import("org.odftoolkit.simple.SpreadsheetDocument")
openAntrag=clzm~loadDocument("antragsmaster.ods")
antragTable=openAntrag~getTableList~get(0)

odfcell=antragTable~getCellByPosition(0,5)
odfcell~setStringValue(tenm)

odfcell=antragTable~getCellByPosition(0,6)
odfcell~setStringValue(tenmsf)

odfcell=antragTable~getCellByPosition(0,11)
odfcell~setStringValue(intern)

odfcell=antragTable~getCellByPosition(2,14)
odfcell~setStringValue(titel)

odfcell=antragTable~getCellByPosition(2,15)
odfcell~setStringValue(surname)

odfcell=antragTable~getCellByPosition(2,16)
```

```
  odfcell~setStringValue(firstname)

  odfcell=antragTable~getCellByPosition(2,17)
  odfcell~setStringValue(postcode town street)

  odfcell=antragTable~getCellByPosition(2,18)
  odfcell~setStringValue(birthday)

  /* add a picture */
  odfcell=antragTable~getCellByPosition(3,22)
  odfcell ,
  ~setImage(.bsf~new("java.net.URI",fotoname))

  odfcell=antragTable~getCellByPosition(2,29)
  odfcell~setStringValue(street)

  odfcell=antragTable~getCellByPosition(2,30)
  odfcell~setStringValue(postcode town)

  odfcell=antragTable~getCellByPosition(2,31)
  odfcell~setStringValue(tel)

  odfcell=antragTable~getCellByPosition(1,33)
  odfcell~setStringValue(Date())

  odfcell=antragTable~getCellByPosition(2,33)
  odfcell ,
  ~setImage(.bsf~new("java.net.URI",signaturname))

  /* save the document */
  openAntrag~save("antrag "firstname surname".ods")
end

/* show message what happened */
.bsf.dialog~messageBox(x" Anträge gespeichert!", ,"information")

return

/* ############################################################## */
/* Rexx event handler "pruefung" */
::class RexxPruefungEventHandler
::method actionPerformed
  use arg eventObject,slotDir
```

```
  call pdf slotDir~userData

/* ############################################################## */
::routine pdf

call lastrow
x=result-1

/* open the master document */
clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
pruefung=clz~loadDocument("pruefungsmaster.ods")
pruefungTable=pruefung~getTableList~get(0)

/* open the import file */
clzs=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
kunden=clzs~loadDocument("kunden.ods")
kundenTable=kunden~getTableList~get(0)

do i=1 to x
  /* read necessary data */
  odfcell=kundenTable~getCellByPosition(1,i)
  firstname=odfcell~getStringValue()
  odfcell=kundenTable~getCellByPosition(2,i)
  surname=odfcell~getStringValue()
  odfcell=kundenTable~getCellByPosition(3,i)
  street=odfcell~getStringValue()
  odfcell=kundenTable~getCellByPosition(4,i)
  postcode=odfcell~getStringValue()
  odfcell=kundenTable~getCellByPosition(5,i)
  town=odfcell~getStringValue()

  /* write new information */
  y=i+4
  odfcell=pruefungTable~getCellByPosition(0,y)
  odfcell~setStringValue(firstname)
  odfcell=pruefungTable~getCellByPosition(1,y)
  odfcell~setStringValue(surname)
  odfcell=pruefungTable~getCellByPosition(2,y)
  odfcell~setStringValue(postcode)
  odfcell=pruefungTable~getCellByPosition(3,y)
  odfcell~setStringValue(town)
  odfcell=pruefungTable~getCellByPosition(4,y)
  odfcell~setStringValue(street)
```

```
end

/* save the outputfile */
pruefung~save("pruefung.ods")

call sysSleep 2

/* initialize connection to the AOo server, get its XContext */
xContext = UNO.connect()
url=uno.convertToUrl(directory()"/pruefung.ods")

/* load the calc file hiddenly */
xDesktop = UNO.createDesktop(xContext)
xComponentLoader = xDesktop~XComponentLoader
props = bsf.createJavaArray(.UNO~propertyValue, 1)
props[1]=uno.createProperty("Hidden",box("boolean", .true))
calcComponent = xComponentLoader ,
~loadComponentFromURL(url,"_blank",0,props)

/* convert file to PDF using a PDF export filter
 to be applied in "storeToURL" and get xStorable interface
 create Java array for two PropertyValue entries*/
xStorable = calcComponent~XStorable
props = bsf.createJavaArray(.UNO~propertyValue, 2)
props[1]=uno.createProperty("FilterName","writer_pdf_Export")
props[2]=uno.createProperty("CompressMode",2)

/* create output file name and store it with props to URL */
storeURL = substr(url, 1, lastpos(".",url)) || "pdf"
xStorable~storeToUrl(storeURL,props)

/* close calcComponent */
x_Closeable = calcComponent~XCloseable
x_Closeable~close(.true)

/* show message what happened */
.bsf.dialog~messageBox("Liste gespeichert!", ,"information")

return

/* ############################################################## */
/* Rexx event handler "mail"  */
::class RexxMailEventHandler
```

```
::method actionPerformed
  use arg eventObject,slotDir
  call mail slotDir~userData

/* ################################################################ */
::routine mail

/* define receiver */
empfaenger = "guentermayer04@gmail.com"

/* define sendermailaddress */
sendermail = "mailrexx@gmail.com"

/* set sender mail account properties */
mail=.BSF~new("java.util.Properties")
mail~setProperty("mail.transport.protocol","smtps")
mail~setProperty("mail.smtp.port","587")
mail~setProperty("mail.smtp.host","smtp.gmail.com")
mail~setProperty("mail.smtp.user",sendermail)
mail~setProperty("mail.smtp.password","mailrexx123")
mail~setProperty("mail.smtp.auth","true")

/* The Session class collects together properties
 and defaults used by the mail API's.*/
impsess=BSF.import("javax.mail.Session")
session=impsess~getDefaultInstance(mail)

/* MimeMessage: */
msge=.BSF~new('javax.mail.internet.MimeMessage',session)
msge~setSubject("TEST")
testtext=.BSF~new('javax.mail.internet.MimeBodyPart')
msgbodypart=testtext~setText("Kandidatenliste")

/* Attachment */
filename = uno.convertFromUrl(directory()"/pruefung.pdf")
testatt=.BSF~new('javax.mail.internet.MimeBodyPart')
source=.BSF~new('javax.activation.FileDataSource',filename)
imphand=.BSF~new('javax.activation.DataHandler',source)
testatt~setDataHandler(imphand)
testatt~setFileName("pruefung.pdf")

/* create Mulitpart */
multipart=.BSf~new('javax.mail.internet.MimeMultipart')
```

```
multipart~addBodyPart(testtext)
multipart~addBodypart(testatt)
fileDataSource=.bsf~new('javax.activation.FileDataSource',filename)
narf=fileDataSource~getContentType

/* --> error message: no content!!! */
msge~setContent(multipart,"application/octet-stream")

/* InternetAdresse of receiver */
new= bsf.importClass("javax.mail.internet.MimeMessage$RecipientType")
ia=.bsf~new("javax.mail.internet.InternetAddress",empfaenger)

/* define receiver */
msge~addRecipient(new~to,ia)
sender = .bsf~new("javax.mail.internet.InternetAddress",sendermail)
msge~setFrom(sender)

/* Get a Transport object that implements
this user's desired Transport protcol */
transport =session~getTransport("smtps")
transport~connect("smtp.gmail.com",465,sendermail,"mailrexx123");
transport~sendMessage(msge,msge~getRecipients(new~to))

/* show message what happened */
.bsf.dialog~messageBox("e-mail wurde versendet!",,"information")

return
```

Figure 26: Program code