WIRTSCHAFTSUNIVERSITÄT WIEN
BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:

Automatische Erstellung von Orientierungshilfen für PräsentorInnen in OpenOffice.org Impress

Englischer Titel der Bakkalaureatsarbeit:

Automated Creation of Guideposts & Hints for Presenters in OpenOffice.org Impress

Verfasser:	Dominik Gundacker
Matrikel-Nr.:	0451615
Studienrichtung:	J033 526 Bakkalaureat Wirtschaftsinformatik
Kurs:	1526 Vertiefungskurs VI / Bakkalaureatsarbeit Electronic Commerce
Textsprache:	Englisch
Betreuerin/Betreuer:	ao. Univ. Prof. Dr. Rony G. Flatscher

Ich versichere:

dass ich die Bakkalaureatsarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

dass ich die Ausarbeitung zu dem obigen Thema bisher weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

_____ _____

Datum Unterschrift

Table of Contents

Table of Figures

Table of Snippets

1 Introduction

This chapter will give you a short overview about the content, the structure and the approach of this bachelor thesis.

1.1 Abstract

This bachelor paper will give an introduction to the OpenOffice.org suite, especially the presentation program Impress, and how the scripting language Open Object Rexx can help to automate the use of it.

The result of the automations should make the life easier for presenters and add some extra value to the presentations.

This paper is divided into two parts. The first part will cover the more theoretical and general part of the bachelor thesis. These chapters will address the technical environment including the OpenOffice.org suite itself, the scripting language Open Object Rexx (ooRexx), the Bean Scripting Framework (BSF) and an overall-view to get familiar with the interaction of those components.

After that, the second part will present some snippets and nutshells that demonstrate the automation of Impress.

The concluding part should give a short summary of the paper.

1.2 Research Question

How can Open Object Rexx help to automate the use of Impress and assist the presenter to make presentations more appealing and attractive for both, the audience and the presenter himself?

1.3 Keywords

OpenOffice.org, Impress, Open Object Rexx, Bean Scripting Framework for Open Object Rexx, Automation, Guideposts, Hints

2 General Part

This chapter contains an overview of all the used tools, applications and frameworks, which are needed for the automation of Impress and other OpenOffice.org applications. In addition to the technical prerequisites, the most important terms will be defined here, to ease the access to the topic of the bachelor paper.

2.1 Open Object Rexx

The paper starts with some introductory words about the scripting language Open Object Rexx and how it emerges from IBM's REXX language. There will also be a section with the basic syntax of Open Object Rexx, which is really not that hard to learn and understand.

2.1.1 History

Open Object Rexx has its seeds in the "human centric language" REXX (**R**estructured E**X**tended E**X**ecutor") implemented by Mike Cowlishaw of IBM between 20 March 1979 and mid-1982. The main purpose of the development of REXX was to replace IBM's mainframe batch language EXEC II.
[Flat06a]

Eventually in 1996 REXX was been standardized by ANSI, which published it under the code ANSI X3.274–1996 "Information Technology – Programming Language REXX".
[WikiRe07]

In the 1990s, there were two newer variants of REXX, which were released:

- NetRexx – this version compiles to Java byte-code and makes use of the Java object model. Therefore it is not generally upwards-compatible with the 'classic' REXX.

- Object Rexx – the generally upwards-compatible version of REXX, which implements the object-oriented paradigm.

[WikiRe07]

As noted above, Object Rexx is the object-oriented approach to the scripting language Rexx. One of the requirements of this version was the full compatibility with other interpreters that were not object-oriented or other Open Source Rexx interpreters currently available.
[OORexx07b]

This was a request of the SHARE SIG (special interest group).
In 1997 after 9 years of development, the commercial version of Object REXX was released included in “OS/2 Warp”. Later on, IBM built versions for Windows, AIX and experimental ports for Linux and Solaris.
[Flat06a]

The last step of REXX becoming Open Object Rexx was the decision from IBM to make their product available to the open source community. They choose the RexxLA for managing the project in 2004.

The first release of Open Object Rexx was announced in March 2005.
[OORexx07a]

2.1.2 RexxLA

“The Rexx Language Association (RexxLA) is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language“ [RexxLA07]

This special interest group was involved in porting Object Rexx to an open source project in 2004 as mentioned above. Every year, the group holds a annual meeting called „Annual International Rexx Symposium“ where the members and other experts present new and interesting projects.

2.1.3 Overview

This paragraph deals with the main features and fundamental principles of the Rexx language. One of the major advantages of Rexx is that it is very easy to learn, even without advanced programming skills, and that it facilitates a rapid development

process.
The main features are the following:

- **Natural language syntax** – With Open Object Rexx writing and maintaining programs is very easy. Intuitive instructions, which are oriented to the English language, make programming accessible to IT or business users. Because Rexx does not use any strange abbreviations, it is easy to read a program or script and readily understand its functionality, for example. ‘System.out.println(“xx”)’ in Java and Say “xx” in Rexx.

- **Object orientation**– Supports effective componentization and all the other advantages of the object-oriented paradigm.

- **Implicit data typing**– In Rexx, there is no need to explicitly declare variables. That reduces the complexity of application code and makes programming more intuitive and faster.

- **Decimal arithmetic** – Open Object Rexx performs calculations in a more accurate way. The absence of rounding makes it easy to comply with legal requirements for financial reporting.

- **Cross-platform interoperability** – Reduces development costs and supports skills transfer across platforms.

- **Rapid diagnostics** – In the case of exceptions and errors, Open Object Rexx provides the user with clear messages and a built- in, multi-level debugger.

[OORexx07a]

2.1.4 Syntax

In order to understand the examples and snippets in the following sections of this paper, a short introduction into the Open Object Rexx language is inevitable. This paper does not want to break any conventions for describing a programming language, so the first piece of code is the infamous „Hello World“ example.

```
1 /* The infamous
2 Hello World ;-) */
3 SAY "Hello World!"
4 EXIT 0
```

Snippet 1: Hello World

This snippet shows many aspects of Rexx:

- **Multi line comments** – Starting with a slash and an asterisk, a comment can span over multiple lines. An asterisk and a slash closes the comment.
- **Single line comment** – Starting with a double minus sign or dash, a comment can be made only till the end of the line.
- **Command end** – Normally, a command consists of all characters till the semicolon. It is possible to write as many commands as one prefers in one line. If there is no semicolon, the command ends at the end of the line.
- **Basic commands** – SAY prints the following string literals, variables, etc. to the command shell where the Rexx script was started.
  EXIT 0, as the name implies, exits the program and sends 0 back to the shell. Whereas 0 generally means that the program finished with no problems or errors.

[Flat06b]

```
 1 a = 1
 2 b = "2"
 3 SAY a  b
 4 SAY a b
 5 SAY a ||b
 6 SAY a + b
 7
 8 OUTPUT:
 9 1 2
10 1 2
11 12
12 3
```

Snippet 2: Variables

This snippet shows the following facts:

One do not need to specify a data type or something similar. Rexx automatically detects if the value inside a variable is a string or a numeric value and handles the

operation itself.

Rexx also treats several white spaces in commands and string concatenations as one whitespace. That is why the output of the *SAY* number 1 and *SAY* number 2 are equal.

The double-pipe in a string concatenation avoids the single whitespace between the strings and connects the value of *a* and the value of *b* seamless together.

The last *SAY* command adds the two values regardless the fact that *b* was initialized with a string literal containing the number 4.
[Flat06b]

The next snippets show how control structures are implemented in Rexx:

```
1 IF month > 6 THEN
2   DO
3     SAY "Welcome to the second half of the year!"
4     SAY "Lets hope it will be as good as the first one..."
5   END
6 ELSE
7   SAY "I think the current month is not october!"
```

Snippet 3: Control Structures 1

The *IF – ELSE* implementation of Rexx looks like in any other programming language. If one wants to make a block of commands, for example to be processed in the *IF* or *ELSE* branch, then the *DO – END* commands are appropriate. There is no need for a enclosing tag.

```
 1 DO 2
 2   SAY "Heho"
 3   SAY "Lets go!"
 4 END
 5
 6 OUTPUT:
 7 Heho
 8 Lets go!
 9 Heho
10 Lets go!
```

Snippet 4: Control Structures 2

With the keywords *DO* and *END* respectively a loop is implemented in Rexx. One can specify the number of iterations after *DO*.

```
 1 DO i = 1 TO 3 BY 0.5
 2   SAY "Value: " i
 3 END
 4
 5 OUTPUT:
 6 Value:  1
 7 Value:  1.5
 8 Value:  2.0
 9 Value:  2.5
10 Value:  3.0
```

Snippet 5: Control Structures 3

The equivalent to a for – loop in Java is the *DO–TO–BY* loop. With this loop one can specify the start value of a control variable, the value when the loop should be exited and the value that would be added to the control variable after each iteration.

```
 1 i = 0
 2 DO WHILE i < 3
 3   SAY "Value: " i
 4   i = i + 1
 5 END
 6
 7 OUTPUT:
 8 Value:  0
 9 Value:  1
10 Value:  2
```

Snippet 6: Control Structures 4

The last snippet shows the *DO-WHILE* loop. In fact it is nearly the same like the *DO-TO-BY* loop, but unlike this version, one has to manage the increase of the control variable himself. The keyword *WHILE* can be changed to *UNTIL*, which alters the way how the loop will be exited.

[Flat06b]

2.2 OpenOffice.org

This paragraph deals with the next essential part of this bachelor paper: The OpenOffice.org office suite. The first part describes the historic facts of the project, later on the paper shows the main components of OpenOffice.org.

OpenOffice.org is a free suite of office applications available under the GNU Lesser General Public License (LGPL). The software package includes a word processor, a spreadsheet, a presentation program, a database program and a vector graphics editor. The suite is targeted to reduce the market share of Microsoft's Office and enjoys increasing popularity amongst the community.
[WikiOo07]

2.2.1 History

The origin of OpenOffice.org is the company StarDivision, which was founded in Germany in 1986 by Marco Börries. OpenOffice.org emerged from the earlier proprietary software application suite StarOffice.
[WikiOo07]

After Sun Microsystems acquired StarDivision in 1999, they released StarOffice 5.2, the first version which was free of charge. [OpenOf07a]

In 2000 Sun released the source code under both the LGPL and the Sun Industry Standards Source License (SISSL). Since the autumn of that year, the product is now called OpenOffice.org. Years later Sun changed the systems with two different license types and decides to continue their work with the usage of the LGPL.

At the time when this paper was written, the current version of OpenOffice.org was 2.2.1 RC1 build SRC680_m16 and was released on May 5th, 2007.
[WikiOo07]

2.2.2 Overview

The OpenOffice.org office suite consists of different applications. Nearly all of them have corresponding applications in Microsoft's Office. This fact should make it easier for users, to switch to the open source suite.

The components are the following:

- **Writer** – The word processing application, similar to MS Word, can be used to create text documents, from simple letters to books or thesis like this one.
- **Calc** – The spreadsheet program is the counterpart to MS Excel with the basic functionalities like tables, cell and calculations but also provides the user with a comprehensive range of advanced functions.
- **Impress –** The presentation software of OpenOffice.org. Like PowerPoint, Impress can be used for creating multimedia presentations. This component will be heavily used across the paper.
- **Base –** Base is a database application and can be compared with MS Access. In Base creation of tables, reports, queries and forms is possible.
- **Draw –** Draw represents a vector graphics editor to create and design everything from simple diagrams to dynamic 3D illustrations.
- **Math –** A tool, which can be used to create mathematical equations, either with a graphical user interface (GUI) or a equation editor.

[WikiOo07] [OpenOf07b]

2.2.3 Architecture

This section will cover the architecture of OpenOffice.org to act as an entry point for the automation. The facts presented here are essential for the understanding of the snippets and nutshells in this paper.

OpenOffice.org is based on a client-server architecture. The communication between the layers is typically supposed to run over TCP/IP sockets by using UNO remote protocol (urp), whereas a typical installation of this office suite runs on a single PC rather than on different machines with each running different operating systems, which is possible.

Figure 1 illustrates the communication between UNO objects.

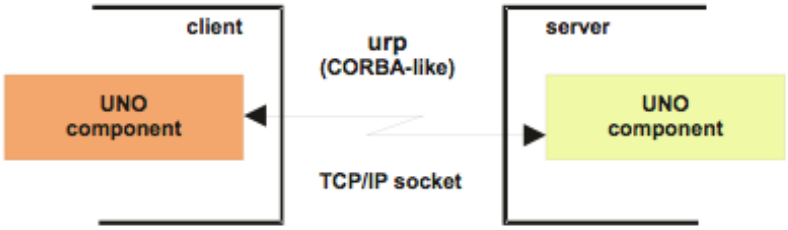

Figure 1: Communication between UNO components [Flat06]

The object model used in OpenOffice.org is called Universal Network Objects (UNO) and every component or object is defined using an interface description language (IDL). That means that each application, for example Impress, represents a set of those components assembled together.

Figure 2 tries to picture this concept, whereas it should be noted that swriter or scalc can here be replaced by any otherOpenOffice.org applications.
[Flat05]

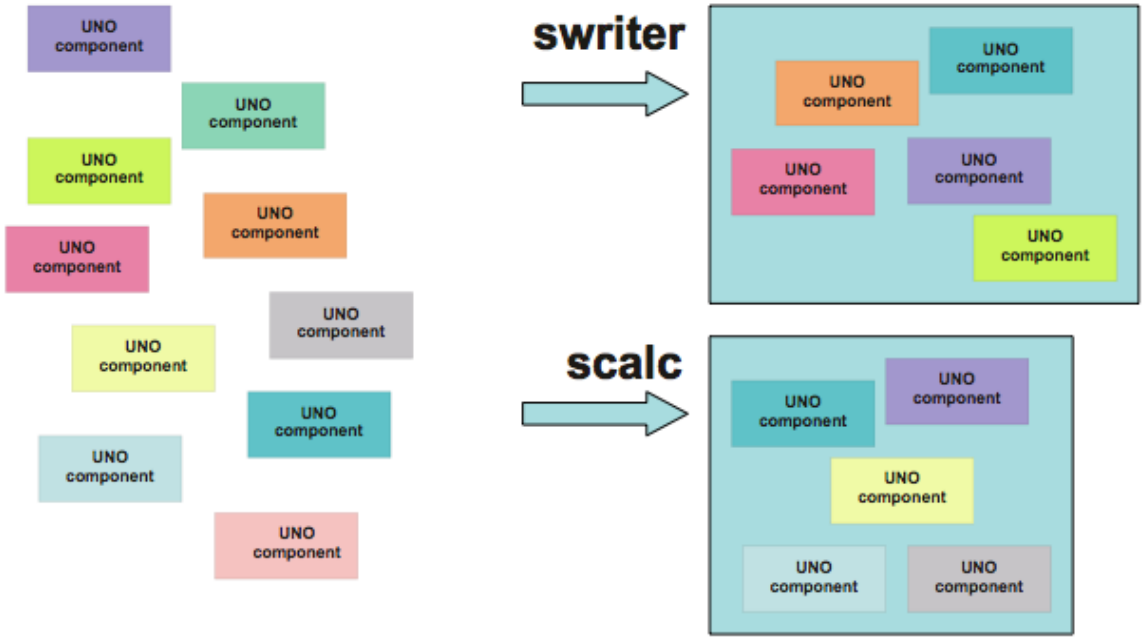

Figure 2: UNO Component Model [Flat05]

2.2.4 Services & Interfaces

Objects

„In UNO, an object is a software artifact that has methods that you can call and attributes that you can get and set. Exactly what methods and attributes an object offers is specified by the set of interfaces it supports.“
[OpenOf07c]

Interface

„An interface specifies a set of attributes and methods that together define one single aspect of an object“
[OpenOf07c]

Each UNO component consists of interfaces and properties, which provide access to the different functionalities of the components. Properties are used for storing information for these services.

To retrieve a new instance of a service component one has to use a ServiceManager, which is an implementation of the factory method pattern in software programming. The methods "*createInstance*()" or "*createInstanceWithArguments*()" in combination with the fully qualified name of the UNO component can be used to create the so-called service objects.

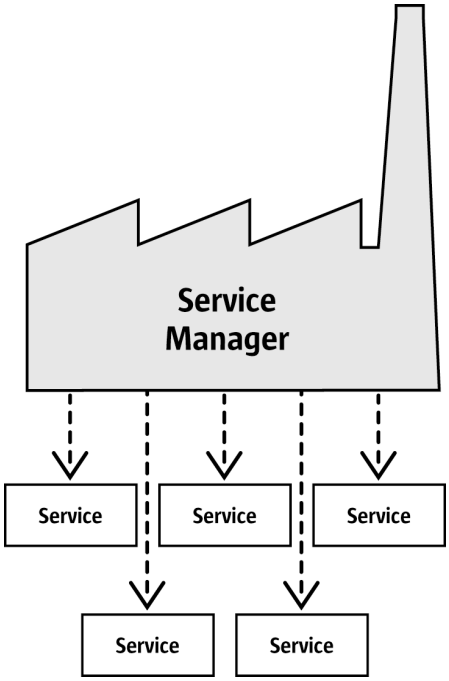

Figure 3: Illustration of ServiceManager [OpenOf07c]

In order to get access to methods from an interface, one has to query the service object for the interface itself. This step is mandatory and returns an object, which has the methods of the requested interface, which can be invoked now.

This work flow seems very awkward, but offers few advantages like the separation and grouping of methods that belong semantically and functionally together.
[Flat05]

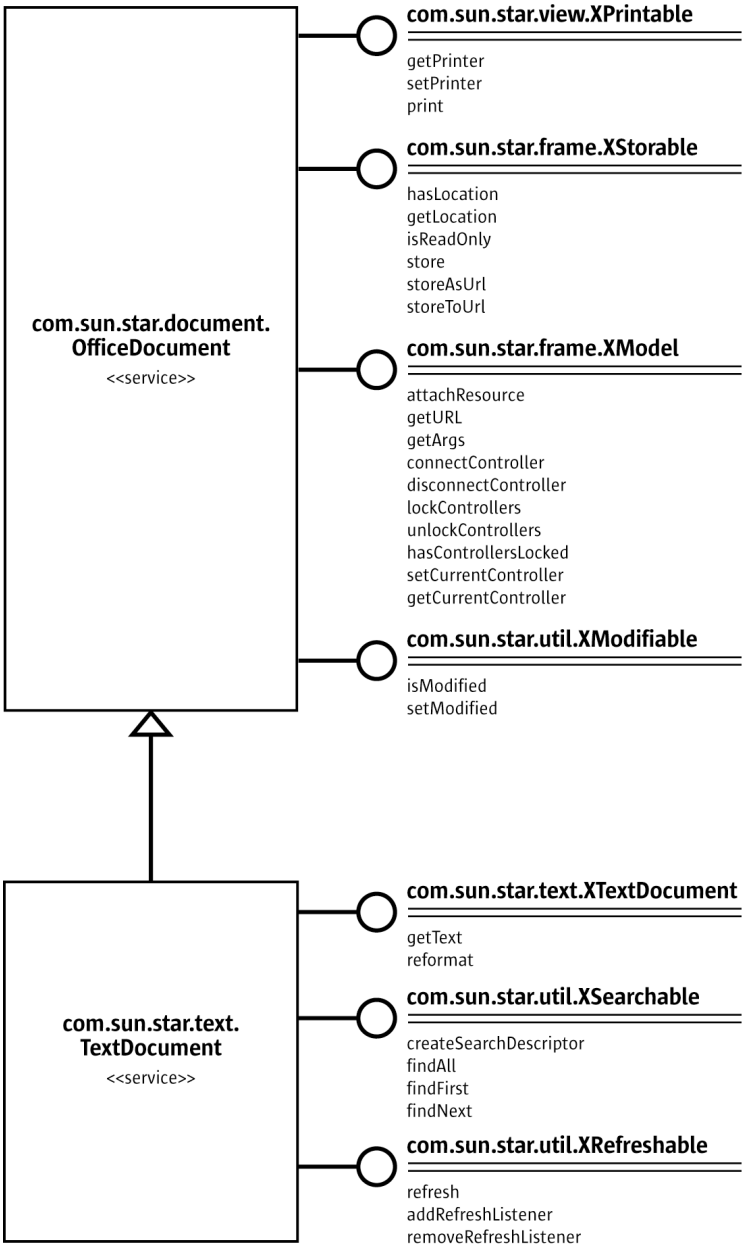

Figure 4: Interfaces & Methods [OpenOf07c]

Figure 4 shows how two objects with interfaces and corresponding methods look like in OpenOffice.org using an UML diagram. *TextDocument* contains text, is searchable and refreshable. Because TextDocument is always an *OfficeDocument*, it implements also the interfaces *XModifiable*, *XModel*, *XPrintable* and *XStorable*. All Interfaces begin with an X in OpenOffice to distinguish them from other entities.

2.3 Bean Scripting Framework for Rexx

The next chapter covers the last step to be able to automate OpenOffice.org using Rexx. Sun provides each UNO Object in OpenOffice.org with a Java adapter so that it can fully be controlled and automated using Sun's own popular programming language. So there is only one link left between Java and Rexx, and Open Object Rexx respectively. The Bean Scripting Framework and BSF4Rexx make this possible.

2.3.1 Bean Scripting Framework

The Bean Scripting Framework provides a bunch of classes, which make it possible to access Java objects and methods from scripting languages. It also provides scripting language support within Java applications.

Especially the first application area is used for automation of OpenOffice.org when we want to access the Java Interfaces of the UNO objects and call the methods via Rexx.
[BSF07a]

Again, IBM was the founder of this project in 1999 and chose the Apache Software Foundation to manage the source code. BSF is now part of the Apache Jakarta Project where it is released under the Apache License.

When BSF was available in version 2.3, it was donated to the Apache Software Foundation, the current version is 2.4.0.
[WikiBS07], [BSF07b]

The supported script languages of BSF are:

- **Javascript** (using Rhino ECMAScript, from the Mozilla project)
- **NetRexx** (an extension of the IBM REXX scripting language in Java, mentioned above)
- **Python** (using Jython)
- **Tcl** (using Jacl)
- **XSLT Stylesheets** (as a component of Apache XML project's Xalan and Xerces)

Some languages are also supported because they have their own BSF engine:

- **Java** (using BeanShell, from the BeanShell project)
- **Groovy**
- **JLog** (PROLOG implemented in Java)
- **JRuby**
- **JudoScript**
- **ObjectScript**
- **Open Object Rexx**, using BSF4Rexx

[BSF07a]

There are two important components in BSF:

- **BSFManager** – This class is responsible for all the registered scripting execution engines and maintains the object registry, which permits scripts access to Java objects.
- **BSFEngine** – Through this interface it is possible to handle script execution and object registration in a generic way, because it provides an abstract view of the scripting language's capabilities.

[BSF07c]

2.3.2 BSF4Rexx

BSF4Rexx, as above mentioned, is an extension to make it possible to use the scripting language Rexx to dive into the world of Java. This means that one can access every Java object or method via Rexx. Also the reverse way, using Rexx in Java, is provided by BSF4Rexx
[BSF4Re07c]

With this bridge, Rexx can get access to the largest external function package on earth, which was additionally ported to each and every important operating system and

hardware platform
[Flat06c]

From the historical point of view, it was Prof. Mag. Dr. Rony G. Flatscher who developed BSF4Rexx in 3 different steps. Each version was named after the city where he was working at a university.

The „Essener Version“ was developed in 2000 in cooperation with a student from Prof. Flatscher, Peter Kalendar. He presented this version in spring 2001 to the RexxLA.

The next Version was called „Augsburger Version“ and was finished in 2003. The major changes, beside some bugfixes, was the addition of external functions from Rexx into the „BSF4Rexx.dll“ package.

The current version, the „Vienna Version“, with the number 2.6 allows Open Object Rexx programs to address Java fields as if they were Open Object Rexx attributes. There are also two important methods, *box* and *unbox*, which can wrap and unwrap primitive data types in Java to the corresponding classes.
[Flat06c], [BSF4Re07c]

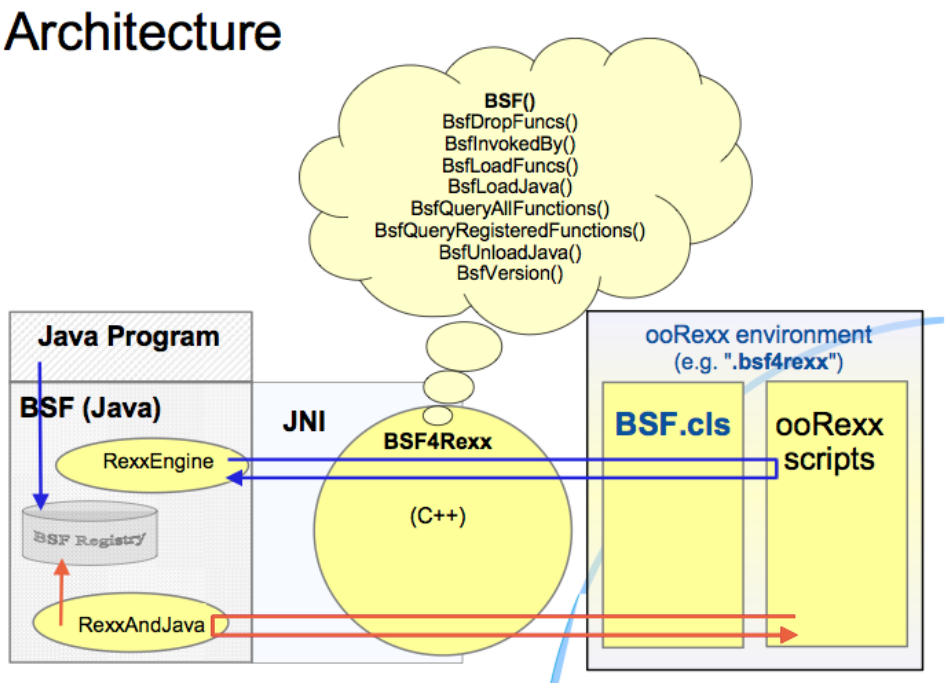

Figure 5: Architecture of BSF4Rexx [Flat06c]

Figure 5 shows the Architecture of the „Vienna Version“ of BSF4Rexx. There is a cls file, which supports BSF routines and contains services that make the most important Java classes directly available via the environment symbol **.bsf4rexx**.
[Flat06c]

2.3.3 BSF.CLS

This section will show how Rexx can access some well known Java classes and how it can call methods and retrieve results from them.

As mentioned above, with the help of the *BSF.CLS* module, one can access a huge amount of classes and functionality, which is available in and for the world of Java. From accessing the standard Java classes, which were shipped with the Java SDK itself like the system properties, date objects, JDBC connections, Swing and SWT GUIs through more sophisticated third party tools and packages. Every Java API can be accessed from the scripting language Open Object Rexx.

The module provides the developer with some basic and general functions for loading classes, for example the method *bsf.loadClass,* which acts as a creator for Java objects. The following snippet should show how easy it is to access a Java Class with just a few lines of code. One remark in advance, in Rexx one can include a module with functions in two different ways:

- **Using ::REQUIRES directive** - This kind of including the module has to be in the last line of your script. It is invoked before any other statement is interpreted and because of this, the functions would be available across the whole Rexx scripts. The next snippet uses this style of import.

- **Using CALL** – The invocation of *CALL* has to be before the first usage of a method of the included module. In the case of including *BSF.CLS*, *CALL BSF.CLS* should be written, before one calls a BSF function. The latter of the snippets for BSF4Rexx uses this style of import.

[Flat06d]

```
1 dateObject = .bsf~new("java.util.Date")
2 stringDate = dateObject~toString()
3 SAY "Todays date is: " stringDate
4 ::requires BSF.CLS
```

Snippet 7: BSF4Rexx simple example

This snippet shows how to create a new *Date* object and store this object into a Rexx variable. *dateObject* contains now a Java object, to which a message is send using the twiddle operator. Sending a message in Rexx means to call a method on this object. Again the result, a *String* object, is saved into a Rexx variable and is printed with the help of the *SAY* command.

The *BSF.CLS* module also contains some useful helper classes, like message boxes and input boxes which are build open the Java class JOptionPane. The next snippet uses a message box to notify the user about something.

```
1 CALL BSF.CLS
2 .bsf.dialog~messageBox("This is a warning! Don't click!!!")
3 SAY "Have you clicked?!"
4 EXIT 0
```

Snippet 8: BSF4Rexx message box example

The script sends the class *dialog* the message *messageBox* with the text that should be displayed. There are some other parameter, which are optional, for example the title of the dialog and the type.

2.3.4 UNO.CLS

This section will present the final step to automate OpenOffice.org programs using the module *UNO.CLS*, which makes the life of developer easier, because this module provides common functionality, which is used across the automation process.

Without using *UNO.CLS* it would take up to 25 lines of code to just open a blank Writer document. All the initialization of *URLResolvers*, *NamingServices* and *Factories* is a very intricate way of doing such basic tasks. And because the process of creating macros or scripts for automation in OpenOffice.org should be a very simple thing to do, *UNO.CLS* provides the developer with a load of helper functions. The difficult parts and those parts, which will be the same for every script one wants to write, are summed up in single methods like *UNO.createDesktop()*. This method takes care of nearly

everything from creating the UNO runtime to instantiating the URL where OpenOffice.org is listening on.

The next snippet will show how easy it is to create an empty Writer document when you are using the *UNO.CLS* module:

```
1 oDesktop = UNO.createDesktop() -- get the OOo Desktop service object
2 xComponentLoader = oDesktop~XDesktop~XComponentLoader - creating Loader
3 -- open an empty .sxw - file
4 xWriterComponent = xComponentLoader -
5 ~loadComponentFromURL("private:factory/swriter",- "_blank", 0, .UNO~noProps)
6
7 ::requires UNO.cls -- get UNO support
```

Snippet 9: Using UNO.CLS

Snippet 9 uses another helper routine contained in the *UNO.CLS*, *.UNO~noProps*. This function creates an empty *Property* object.

Because *UNO.CLS* already includes *BSF.CLS*, there is no need for a *REQUIRE* statement in those scripts, which want to use the OpenOffice.org UNO modules.

2.3.5 Routines

A short overview of some methods presented in the reference card included in the current BSF4Rexx version:

- *uno.createDesktop([context]) - returns the local OpenOffice desktop object*
- *uno.getProperties(o) - returns a blank delimited, encoded string with all defined properties for the service object o*
- *uno.getScriptContext() - returns a UNO proxy, if the ooRexx script was invoked by OpenOffice, .nil else. The UNO proxy object has the following methods, returning context related UNO proxy objects:*
    - *getDocument (the document service object, an XModel)*
    - *getDesktop (the desktop service object, an XDesktop)*
    - *getComponentContext (the context object, an XComponentContext)*

[BSF4Re07a] [BSF4Re07b]

2.4 Summary

This paragraph sums up the previous chapters to make again clear, how the automation of Impress in this bachelor paper is going to be accomplished.

To illustrate this procedure Figure 6 from a previous bachelor thesis from a student of the university of business administration in Vienna shows the different layers between a method call in Open Object Rexx and OpenOffice.org.

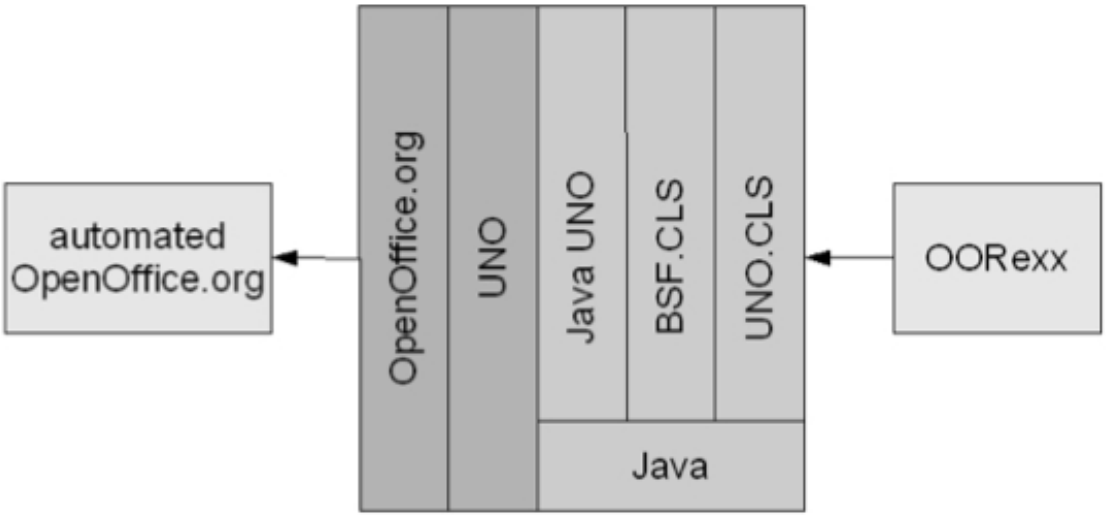

Figure 6: From ooRexx to Automation [Aham05]

The scripts developed in Open Object Rexx are using the module *UNO.CLS* to benefit from the simplified object creations and initializations. In the next step, the procedures in *UNO.CLS* use the functionality provided by the module *BSF.CLS*. As written in the chapter about BSF, one can use a scripting language, in this case and with the BSF4Rexx extension the scripting language Open Object Rexx, to make use of Java.

Exactly this point brings substantial improvement to the development process because

- it is not necessary to develop in Java and
- by the use of *UNO.CLS* one can save a lot of lines of code

In OpenOffice.org there is a comprehensive support of Java to communicate with the UNO components. This represents the Java UNO layer on Figure 6.

The Java UNO model then is connected to the UNO certainly and this leads to OpenOffice.org itself to conduct the automation.
[Flat05]

As the last chapters have shown, an open source scripting language can be used to control and automate an open source office suite on behalf of OpenOffice.org. But not only that, OpenOffice.org itself can invoke Open Object Rexx macros.

This fact is very important, because all of the snippets for automating Impress will be written in form of macros.

It combines the power and straightforwardness of Open Object Rexx with the full featured and open interfaces of OpenOffice.org to make it possible to easily automate the programs of this office suite, even without wide experience in the field of programming and development.

3 Installation Guide

This chapter will describe and show how to set up the environment for developing and running the snippets provided by this paper. This section will only talk about the current versions of each application and programming language.

The steps 3.1, 3.2. and 3.3 must not be followed by in this specific order. The only step, which has a fixed order is step number 4, the installation of BSF4Rexx. The other installations should be finished prior to this task.

3.1 Installation Java

First of all, you have to have at least Java 1.4 installed on your machine to work with BSF4Rexx. Because it is enough to have the Java Runtime Edition (JRE) installed nearly every PC is capable for automating OpenOffice.org from the Java point of view.

But you can also use the Java Development Kit (JDK), which has the JRE already bundled with it, for development.

To check whether the computer has a Java installation at all or if the Java version can be used for the snippets, the command *java –version* in the command prompt will show the result.

You can download Java from Sun's web page at http://java.sun.com. It is also very useful if you download or at least use the API documentation available at http://java.sun.com/reference/api/. Here you can choose the proper documentation with the version number you are using.

The current version of Java is Java 6 (JDK 6u1) and is available at http://java.sun.com/javase/downloads/index.jsp. The version used for developing the nutshells in this bachelor paper is 1.6.0_01.

[BSF4Re07c]

3.2 Installation OpenOffice.org

You can download the latest version of OpenOffice.org at http://download.openoffice.org/index.html. If the desired PC has no valid Java installation there is an option on the download page to include the Java JRE with the download. For all of the people who skipped the latter paragraph, this option would be applicable.

The current version of OpenOffice.org is 2.2.1, which is used for writing this paper.

You can download the office suite for Windows, Linux, Solaris, Mac OS X and Free BSD and in different languages. The automation itself was only tested in Windows but it should also work with Linux.

3.3 Installation Open Object Rexx

Open Object Rexx can be downloaded from the website http://www.oorexx.org/download.html.

Open Object Rexx, like OpenOffice.org is available for different operating systems like Windows, Mac OS X, Linux, AIX and Solaris. Again, the documentation can also be downloaded from the website.

The current version of Open Object Rexx is 3.1.2 and is used for creating the snippets in this bachelor paper.

3.4 Installation BSF4Rexx

The installation of BSF4Rexx is probably the most sophisticated step for preparing the computer for the automation snippets. But if one follows the installation guide supplied with the release and the previous steps were finished successfully, there should be no problems, even for non-experts.

The current release of the BSF4Rexx is available under http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current the web space from Prof. Dr. Mag. Flatscher on the university of business administration web server.

BSF4Rexx comes in form of a zip file, which contains all files, the installation scripts, some examples from previous bachelor thesis' and other Rexx community members and installation guides for every step.

For the first part of the installation, the BSF4Rexx part, the file *readmeBSF4Rexx.txt* is the proper one. To start the installation run the setup script *setup.rex* using the rexx command. This command should now be available in the command prompt after installing Open Object Rexx in one of the previous steps.

This script should create four new files:

- bsf4rexx.cmd – With this, BSF4Rexx scripts can already be run
- installBSF4Rexx.cmd – This script copies the Java archives (jars) and dynamic link libraries to the used Java extension folder. Now every Java application can use BSF4Rexx
- *uninstallBSF4Rexx.cmd* – This script undoes the actions from the install script.
- *setEnvironment4BSF4Rexx.cmd* – This script sets the proper environment variables for using BSF4Rexx

On a Linux PC's all the generated scripts will not have the extension cmd, the scripts are .sh files.

Running the script installBSF4Rexx will finish the installation by now and BSF4Rexx can be tested by calling different kinds of commands to ensure that everything is

working correctly.

Running the scripts *infoBSF.rex* and *infoBSF-oo.rex* with both the *rexx* and *rexxj* command should give the user an output in form of the registered BSF functions.

[BSF4Rexx07-3]

The second part of the installation is to make the bridge to OpenOffice.org and install the Rexx support for the office suite.
The corresponding readme file is called *readmeOOo.txt*.

The first step is to ensure that Java is enabled for OpenOffice.org. For this reason start any application of the office suite, for example Writer, and select the menu item *Options* in the *Tools* menu. In the tree on the left side there is an entry called *Java,* which shows some options on the right side of the dialog.

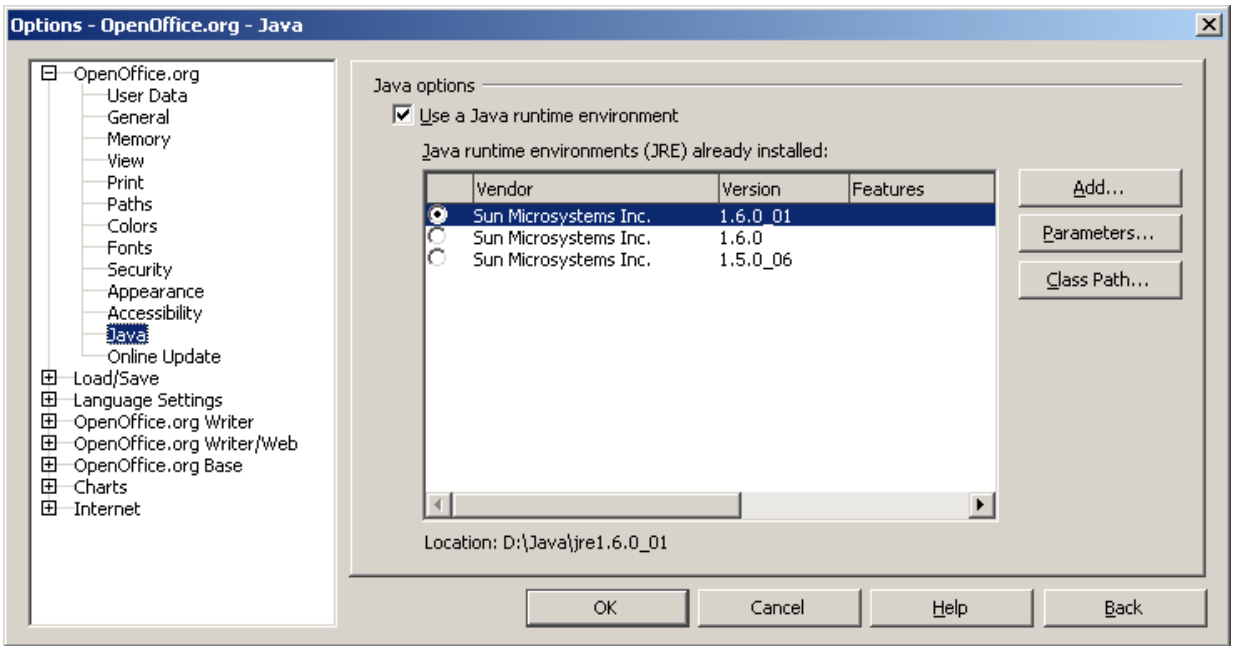

Figure 7: Options Dialog in OOo

Figure 7 shows the dialog in which the user has to select the desired JRE. This version should be the same as the version for which BSF4Rexx is installed. After making changes on this settings, OpenOffice.org as well as the QuickStarter should be closed.

Running the command *rexx setupOOo.rex path-toOOo-directory* will create the install scripts for the OpenOffice.org support.

An example:
rexx setupOOo.rex „d:\OpenOffice.org 2.2“.

Again some scripts for installing the support, uninstalling and setting up of the environmental variables will be generated.
Running the *installOOo.cmd* will actually install the support. All Instances of OpenOffice.org should be closed to ensure that the configuration takes effect in every module.
To test the OpenOffice.org support there are a two Rexx scripts (*testOOo.rex* and *testOOo2.rex*) which should be started with either the *rexx* and *rexxj* command.
The scripts should open a Writer document and insert some text into the document.

The last step for the configuration and installation is the macro support in OpenOffice.org. To do that click on the menu entry „*Extension Manager*“ in the *Tools* menu in an OpenOffice.org application. After clicking on *Add* and selecting the jar-*file ScriptProviderForooRexx.jar*, it will install the macro support for Open Object Rexx.

[BSF4Re07d]

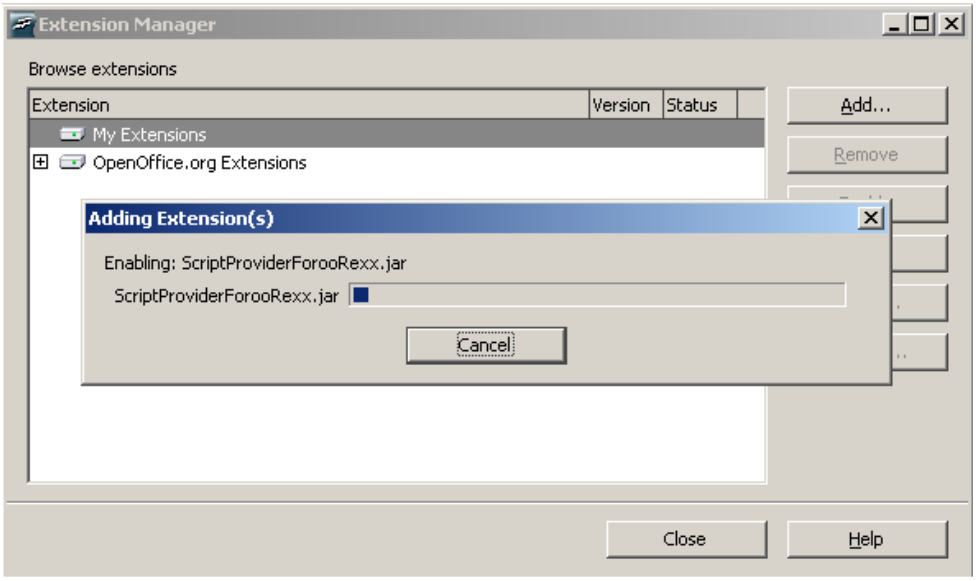

Figure 8: Extension Manager

After restarting OpenOffice.org, in the menu *Macros => Organize Macros* you can see if the installation was finished successfully. Figure 9 shows exactly how it should look like.

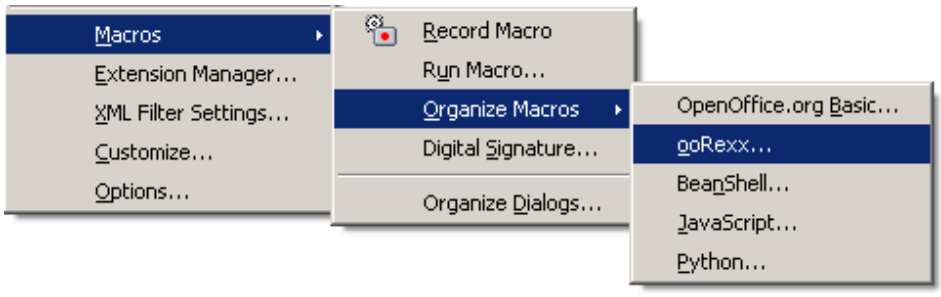

Figure 9: Macros installation

4 Impress Automation

After creating the basis knowledge for conducting automation in OpenOffice.org, this chapter contains general information about the document overview of Impress and some descriptive words regarding the automation.

This bachelor paper wants to show how Open Object Rexx can ease the life of presenters using Impress. The examples presented in this paper can also brush up and advance existing presentations also in consideration of the audience.

A lot of presentations created by students, managers or normal John Does lack of different kinds of gimmicks, which are very useful for both the presenter himself and the audience. Those improvements can be guideposts to illustrate the status of the current slide in the presentation and other helpful and valuable things.

The implementation of this automation will be developed in form of macros written in Open Object Rexx, because this kind of realization would allow running the scripts on existing presentation by selecting a menu entry in the OpenOffice.org application.
This paper will not cover any stand-alone Open Object Rexx scripts because such behavior will not meet the given requirements for this application area.
The starting point of each automation process is an existing (existing from the content point of view) presentation, which should be further enhanced by the macros provided in this bachelor paper.

4.1 General

In OpenOffice.org Impress and Draw are vector-oriented applications for creating presentations and drawings. Both applications support different kind of things to draw, for example rectangles, text, curves and other graphic shapes.

Unlike to the text documents in Writer and the spreadsheet document in Calc, Draw and Impress use the so-called drawpages for displaying the content. Figure 10 reflects this fact with the drawpage container in the middle of the graphic. The document service manager pictured with the fabric symbol at the top of the figure is used for creating all the drawing elements and shape objects. These objects will be later inserted into the drawpages.

There are many different types of shapes to add to a drawpage in Impress & Draw, the most important ones are:

- *GraphicObjectShape*, which can display an image from a file
- *ConnectorShape*, which can connect other *Shapes*
- *EllipseShapes* can be various circles and ellipses.
- *RectangleShape* that pictures a rectangle on a drawpage
- *TextShape* can be used for displaying text on a slide

The controller is used to present the presentation in the GUI and for assigning styles and layouts to the drawings.

The figure also shows the interfaces for accessing the *MasterPages* and the *LayoutManager*.

[OooDev05]

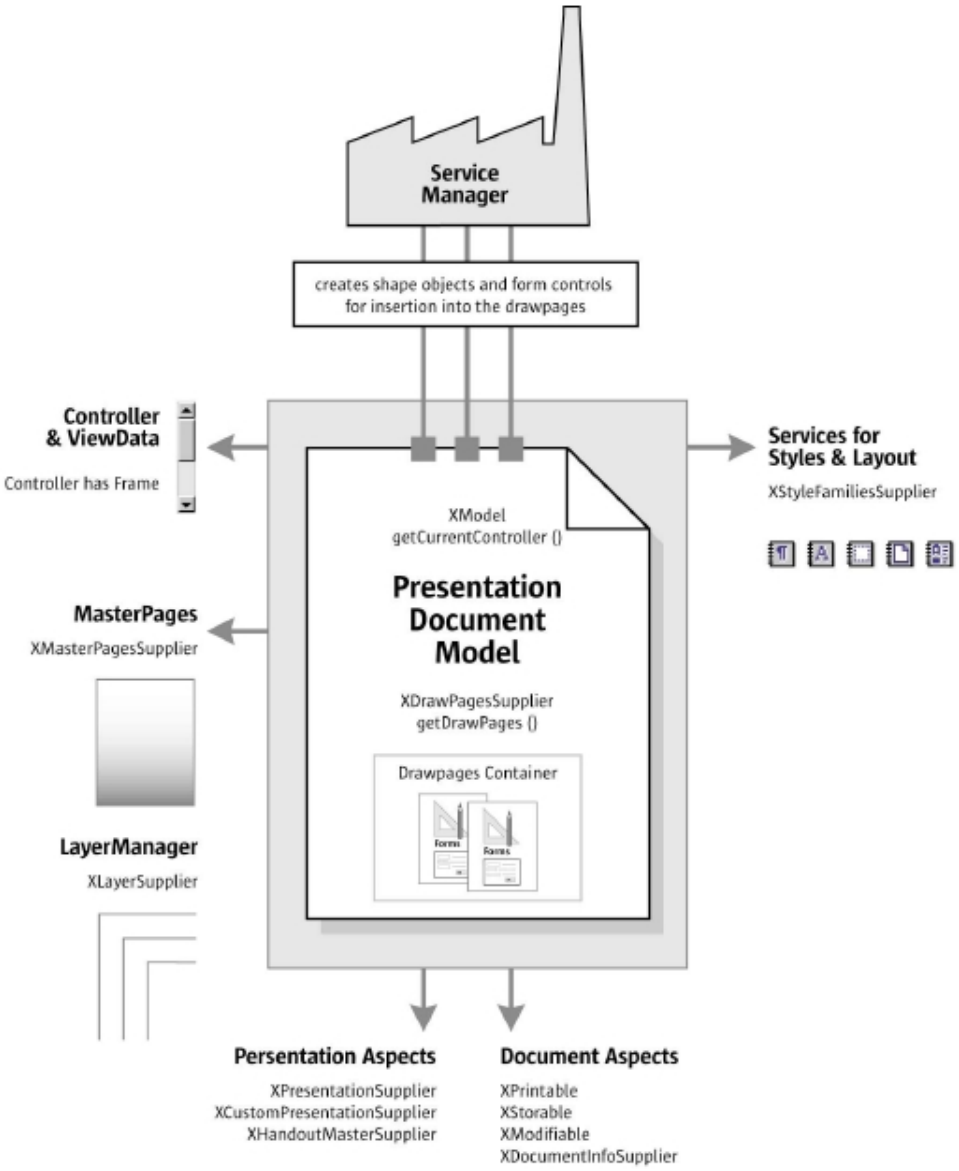

Figure 10: Presentation Document Model [OOoDev05]

4.2 Introduction Example

To make the introduction for the Open Object Rexx macros easier, this paragraph will talk about a very simple macro, which adds a slide to an existing presentation and puts a text field with a type of the notorious „Hello World“ label on each additional slide. The previous slide will get a text field that causes a slide transition to the next page when someone clicks on the text.

There will be an examination of every action taken in the script so that the basic functionality, like getting access to the draw pages, will be cleared by now.

Most of the functions, interfaces and properties will be used throughout the whole bachelor paper.

Because the snippets will be executed as a macro inside OpenOffice.org the first thing to do is to open an existing presentation or to create a new one using Impress.

In the menu *Tools – Macros – Organize Macros* there is the entry *ooRexx,* which opens a dialog to create, edit and run the written macros, as shown on Figure 11,

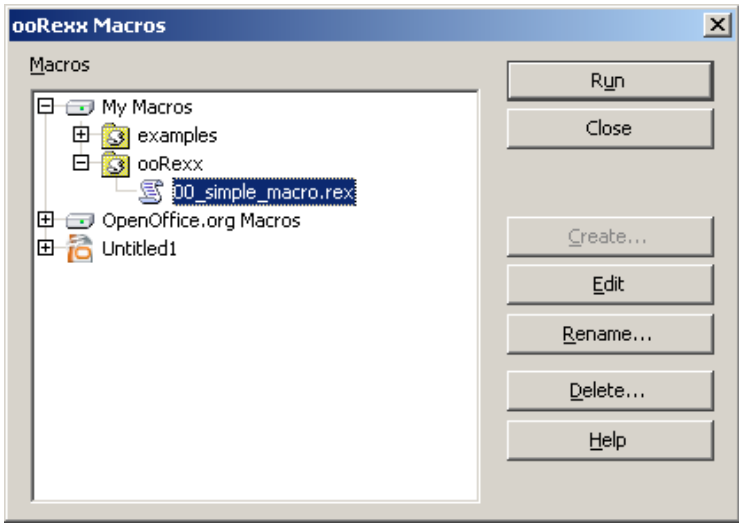

Figure 11: Organize Macros

The dialog for editing the macros acts also as a debug window where the lines, which cause an error are marked and the error message is shown. Unfortunately there is no syntax highlighting in the edit dialog, so the best thing one can do is to write the macros in a third party editor like gvim, which supports the syntax of Open Object Rexx. Then, after writing the scripts, they can be copied into the macro editor of OpenOffice.org.

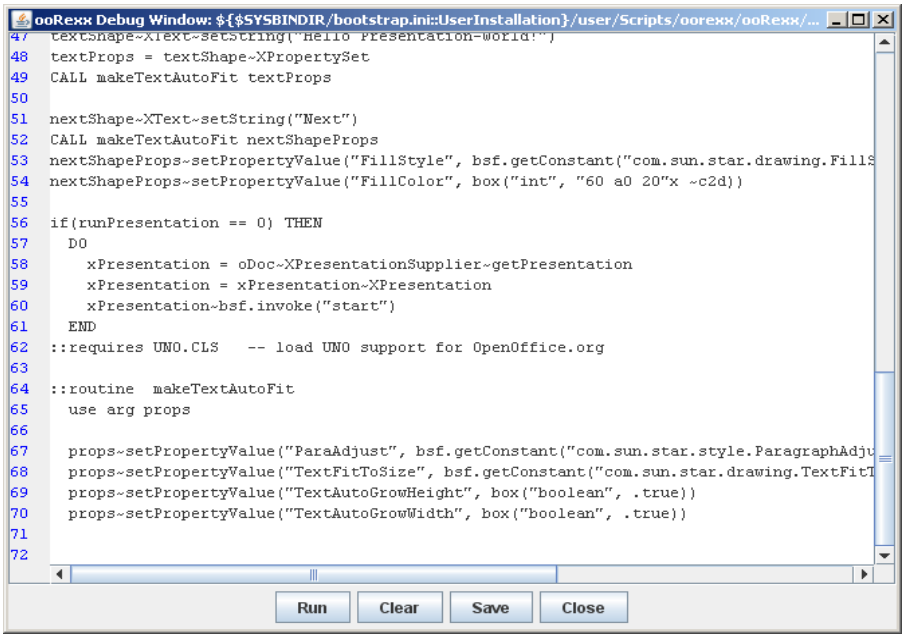

Figure 12: Macro editor

The dialog is shown on the Figure 12, from here the script can be executed. The source code of the introduction snippet can be seen in the Snippet 11.

```
 5 oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
 6 /* retrieving the important interfaces to get access to the drawpages */
 7 runPresentation = .bsf.dialog~dialogbox("Do you want to start the "-
 8  "presentation after adding the new slide?", "Question", "question", "YesNo")
 9 xDrawPagesSupplier=oDoc~XDrawPagesSupplier
10 xImpressFactory = oDoc~XMultiServiceFactory
11 xDrawPages = xDrawPagesSupplier~getDrawPages
```

This cutout from Snippet 11 will get the *ScriptingContext* whose function is to provide the developer with the document services and interfaces.

This is specific for the macros in Open Object Rexx, if one wants to work with the Impress interfaces from a standalone script, the *UNO.createDesktop()* procedure will get the related interfaces.

```
 1 oDesktop          = UNO.createDesktop()     -- get the UNO Desktop service object
 2 xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
 3                                                        -- interface
 4 url = "private:factory/simpress"
 5 xImpressComponent = xComponentLoader~loadComponentFromURL(url,-
 6 "_blank", 0, .UNO~noProps)
 7 xImpressFactory = xImpressComponent~XMultiServiceFactory
 8 xDrawPagesSupplier = xImpressComponent~XDrawPagesSupplier
 9
10 xDrawPages = xDrawPagesSupplier~getDrawPages
```

Snippet 10: Interfaces with Desktop

```
/* Simple Example to demonstrate some basic functionality */
/* 00_simple_macro.rex */
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object

oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
runPresentation = .bsf.dialog~dialogbox("Do you want to start the "-
 "presentation after adding the new slide?", "Question", "question", "YesNo")
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
pagecount=xDrawPages~XIndexAccess~getCount
lastPage = pagecount - 1;
/* inserting a new drawpage (slide) */
xDrawPages~insertNewByIndex(pagecount)
xDrawPageLast = xDrawPages~getByIndex(lastPage)~XDrawPage
xDrawPageNew = xDrawPages~getByIndex(pagecount)~XDrawPage
drawPageProps = xDrawPageLast~XPropertySet
widths = drawPageProps~getPropertyValue("Width")
widths = trunc(widths / 2)

CALL bsf.import "com.sun.star.awt.Size", "awtSize"
CALL bsf.import "com.sun.star.awt.Point", "awtPoint"
/* creating, resizing and positioning of the frist textshape */
textShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
textShape = textShape~XShape
shapeWidth = 10000
shapeHeight = 3000
shapeX = widths - trunc(shapeWidth / 2)
shapeY = 5000
textShape~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
textShape~setPosition(.awtPoint~new(shapeX, shapeY))

/* creating, resizing and positioning of textshape for getting to next page */
nextShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
nextShape = nextShape~XShape
shapeWidth = 9500
shapeHeight = 1000
shapeX = widths - trunc(shapeWidth / 2)
shapeY = 8000
nextShape~setSize(.awtSize~new(shapeWidth, shapeHeight))
nextShape~setPosition(.awtPoint~new(shapeX, shapeY))

/* defining the slide transitions */
xDrawPage1Props = xDrawPageNew~XPropertySet
xDrawPage1Props~setPropertyValue("Change", box("int", 1))
xDrawPage1Props~setPropertyValue("Duration", box("int", 5))

/* adding the shapes to the different drawpages */
xDrawPageLast~add(nextShape)
xDrawPageNew~add(textShape)

/* formating the textshapes */
textShape~XText~setString("Hello Presentation-World!")
textProps = textShape~XPropertySet
CALL makeTextAutoFit textProps

nextShapeProps = nextShape~XPropertySet
nextShapeProps~setPropertyValue("OnClick", -
 bsf.getConstant("com.sun.star.presentation.ClickAction","NEXTPAGE"))
nextShape~XText~setString("Next")
CALL makeTextAutoFit nextShapeProps
nextShapeProps~setPropertyValue("FillStyle",-
 bsf.getConstant("com.sun.star.drawing.FillStyle", "SOLID"))
nextShapeProps~setPropertyValue("FillColor", box("int", "60 a0 20"x ~c2d))

```

```
67 /* start the presentation if asked for it */
68 if(runPresentation == 0) THEN
69   DO
70     xPresentation = oDoc~XPresentationSupplier~getPresentation
71     xPresentation~bsf.invoke("start")
72   END
73 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
74
75 ::routine  makeTextAutoFit
76   use arg props
77
78   props~setPropertyValue("ParaAdjust",-
79    bsf.getConstant("com.sun.star.style.ParagraphAdjust", "CENTER"))
80   props~setPropertyValue("TextFitToSize",-
81    bsf.getConstant("com.sun.star.drawing.TextFitToSizeType","PROPORTIONAL"))
82   props~setPropertyValue("TextAutoGrowHeight", box("boolean", .true))
83   props~setPropertyValue("TextAutoGrowWidth", box("boolean", .true))
```

Snippet 11: Introduction example (00_introduction.rex)

```
25 textShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
```

This cutout of the source code creates a *TextShape* for displaying the text „Hello Presentation-World“ on the new slide.

```
58 nextShapeProps = nextShape~XPropertySet
59 nextShapeProps~setPropertyValue("OnClick", -
60  bsf.getConstant("com.sun.star.presentation.ClickAction","NEXTPAGE"))
```

If a user clicks on the *nextShape TextShape*, the presentation goes on to the next slide. The function *getConstant* from the *BSF.CLS* module is used a lot in the macros. It is an easy way to retrieve constants from classes with the use of the class name and the name of the constant.

```
44 /* defining the slide transitions */
45 xDrawPage1Props = xDrawPageNew~XPropertySet
46 xDrawPage1Props~setPropertyValue("Change", box("int", 1))
47 xDrawPage1Props~setPropertyValue("Duration", box("int", 5))
```

The newly created page properties will be changed here. The property value *Change* specifies how the transition will be conducted. A value of 0 means that the user needs to click to trigger the effects and to change to the next slide. 1 means that the page is automatically switched and 2 means that every effect will run automatically but the user needs to click to change to the next slide.

If *Change* has the value 1, *Duration* specifies how many seconds each page will be shown.

```
49 /* adding the shapes to the different drawpages */
50 xDrawPageLast~add(nextShape)
51 xDrawPageNew~add(textShape)
```

The two shapes, one *TextShape* for the „Hello World“ label and the other one for the click text field to go to the next slide, are added to the drawpages using the add method.

```
67 /* start the presentation if asked for it */
68 if(runPresentation == 0) THEN
69   DO
70     xPresentation = oDoc~XPresentationSupplier~getPresentation
71     xPresentation~bsf.invoke("start")
72   END
73 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
```

The variable *runPresentation* is 0 when the user does want to start the presentation after the changes of the slides were done. The impress document will be queried for the *XPresentationSupplier* interface. This interface has a method to get the *XPresentation* interface to start the presentation.

The next two figures show the output of the macro.

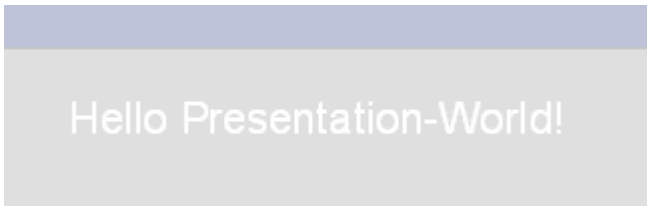

Figure 13: "Hello World" TextShape

Figure 14: ClickEvent on TextShape

After running the script, the presentation will be updated and depending on the decision made at the beginning, it will also be started.

4.3 Examples

After describing the basic functions for automation of Impress with the help of the introduction example, the next paragraph will be dealing with the other examples. These examples will create helpful features and additions to presentations.

This paper will also include some examples, which will take off some recurring tasks from the presenter so that she or he just needs to concentrate on the content of the presentation. The macros will improve the presentations and give the audience a better understanding of the progress of the address by creating guideposts and progress bars.

The first four macros will be dealing with the progress of presentations and give the audience an information, how far the presentation is already advanced.
Furthermore the next two macros will provide the presenter with a possibility to take over recurring tasks. The next two macros are the most sophisticated ones. They will create guideposts on each slide, with the main headings of the presentation. The last example will automatically create the agenda of a presentation.

4.3.1 Example01

```
/* Macro, which generates a progress bar at the bottom of each slide */
/* 01_progressbar.rex */
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* initialize all variables (height, width, etc.) */
pagecount=xDrawPages~XIndexAccess~getCount
firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
width = firstDrawPageProps~getPropertyValue("Width")
height = firstDrawPageProps~getPropertyValue("Height")
shapeWidthBorder = width - 1000
shapeHeight = 750
shapeX = 500
shapeY = height - 1250

IF pagecount == 1 THEN
DO
   .bsf.dialog~messageBox("This presentation has only one slide. "-
     "There is no need for a progressbar!", "ERROR", "error")
   EXIT 0
END

step = trunc((width - 1000) / (pagecount - 1))

currentStatus = step
DO i = 1 TO pagecount - 1
   xDrawPage = xDrawPages~getByIndex(i)~XDrawPage

   /* remove existing bars, if necessary */
   xShapes = xDrawPage~XShapes
   DO j = 0 TO xShapes~getCount - 1
      xShape = xShapes~getByIndex(j)
      IF(xShape~XNamed~getName() == "progressbar_group") THEN
      DO
         xShapeGroup = xShape~XShapeGroup
         xDrawPage~remove(xShapeGroup)
      END
   END

   /* creating and positioning of border of the bar shape */
   barBorder = xImpressFactory~createInstance(-
     "com.sun.star.drawing.RectangleShape")
   barBorder = barBorder~XShape
   CALL setSizeAndPosition barBorder, shapeWidthBorder, shapeHeight,-
     shapeX, shapeY
   xDrawPage~add(barBorder)
   barBorderProps=barBorder~XPropertySet
   fillStyles = bsf.wrapStaticFields("com.sun.star.drawing.FillStyle")
   barBorderProps~setPropertyValue("FillStyle", fillStyles~none)
   CALL setShadowAndFormat(barBorderProps)

   /* creating and positioning of the statusbar shape */
   statusBarShape = xImpressFactory~createInstance(-
     "com.sun.star.drawing.RectangleShape")
   statusBarShape = statusBarShape~XShape
   CALL setSizeAndPosition statusBarShape, currentStatus, shapeHeight,-
     shapeX, shapeY
   currentStatus = currentStatus + step
   xDrawPage~add(statusBarShape)
```

```
 67
 68   /* changing the colors of the filling */
 69   statusBarShapeProps=statusBarShape~XPropertySet
 70   statusBarShapeProps~setPropertyValue("FillStyle",fillStyles~gradient)
 71   CALL CreateGradientObject
 72   statusBarShapeProps~setPropertyValue("FillGradient", result)
 73   statusBarShapeProps~setPropertyValue("LineStyle",-
 74     bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
 75   CALL setShadowAndFormat(statusBarShapeProps)
 76
 77   /* create the group */
 78   shapeGroup = xMcf~createInstanceWithContext(-
 79     "com.sun.star.drawing.ShapeCollection", xContext)
 80   shapeGroup = shapeGroup~XShapes
 81   shapeGroup~add(barBorder)
 82   shapeGroup~add(statusBarShape)
 83   xShapeGrouper = xDrawPage~XShapeGrouper
 84   xShapeGroup = xShapeGrouper~group(shapeGroup)
 85   name = xShapeGroup~XNamed
 86   name~setName("progressbar_group")
 87 END
 88 EXIT 0
 89 /* Function for creating the GradientObject */
 90 CreateGradientObject :
 91   gradient = .bsf~new("com.sun.star.awt.Gradient")
 92   gradient~Style = bsf.getConstant("com.sun.star.awt.GradientStyle", "LINEAR")
 93   gradient~StartColor     = 9282303
 94   gradient~EndColor       = 0
 95   gradient~Angle          = 120
 96   gradient~Border         = 0
 97   gradient~XOffset        = 0
 98   gradient~YOffset        = 0
 99   gradient~StartIntensity = 100
100   gradient~EndIntensity   = 100
101   gradient~StepCount      = 10
102
103   return gradient
104
105 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
106
107 /* routine for positioning and resizing a shape */
108 ::routine  setSizeAndPosition
109   use arg shape, width, height, posX, posY
110
111   shape~setSize(-
112     .bsf~new("com.sun.star.awt.Size", width, height))
113   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
114
115
116 /* routine for setting the shadow*/
117 ::routine  setShadowAndFormat
118   use arg props
119
120   props~setPropertyValue("CornerRadius", box("int", 300))
121   props~setPropertyValue("Shadow", box("boolean", .true))
122   props~setPropertyValue("ShadowXDistance", box("int", 150))
123   props~setPropertyValue("ShadowYDistance", box("int", 150))
124
125 /* routine for removing selection*/
126 ::routine  removeSelection
127   use arg oDoc
128
129   model= oDoc~XModel
130   controller = model~getCurrentController()
131   selectionController = controller~XSelectionSupplier
132   selected = selectionController~getSelection()
133   selectionController~select(.nil)
```

Snippet 12: Progressbar (01_progressbar.rex)

The first example for advancing presentations in OpenOffice.org adds a progress bar to the bottom of each slide. Because in most presentations, the first slide is some sort of a welcome or introduction page, this page will not contain a progress bar.

That means on the second slide you can see a rectangle with a filled section on the left side:

Figure 15: Progress bar

Each slide will show new shapes, where the filled shapes width is increased incremental. This will give the audience a feeling how advanced the progress of the presentation is. Also the presenter will get a critical information about the state of the presentation.

As the presentation goes on, the rectangle will be filled more and more till it reaches the end.

```
33 DO i = 1 TO pagecount - 1
34    xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
35
36    /* remove existing bars, if necessary */
37    xShapes = xDrawPage~XShapes
38    DO j = 0 TO xShapes~getCount - 1
39       xShape = xShapes~getByIndex(j)
40       IF(xShape~XNamed~getName() == "progressbar_group") THEN
41       DO
42          xShapeGroup = xShape~XShapeGroup
43          xDrawPage~remove(xShapeGroup)
44       END
45    END
46
47    /* creating and positioning of border of the bar shape */
48    barBorder = xImpressFactory~createInstance(-
49      "com.sun.star.drawing.RectangleShape")
50    barBorder = barBorder~XShape
51    CALL setSizeAndPosition barBorder, shapeWidthBorder, shapeHeight,-
52      shapeX, shapeY
53    xDrawPage~add(barBorder)
```

The first thing to do is to go through all the existing drawpages. This will be realized with a *DO – TO* iteration starting with the drawpage with index 1, which is actually the second slide because the index is 0 based. If there is already a progress bar, the grouped shape will be removed. The macro then generates a *RectangleShape* and passes it to a routine, which resizes the component and places it to the correct position.

```
107 /* routine for positioning and resizing a shape */
108 ::routine  setSizeAndPosition
109   use arg shape, width, height, posX, posY
110
111   shape~setSize(-
112     .bsf~new("com.sun.star.awt.Size", width, height))
113   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
```

This snippet shows the routine for positioning and resizing shapes, which is used in every macro. As one can see the routine creates a *Size* Object with arguments number two and three and a *Point* Object with the latter one. Argument number one is the shape itself.

```
116 /* routine for setting the shadow*/
117 ::routine  setShadowAndFormat
118   use arg props
119
120   props~setPropertyValue("CornerRadius", box("int", 300))
121   props~setPropertyValue("Shadow", box("boolean", .true))
122   props~setPropertyValue("ShadowXDistance", box("int", 150))
123   props~setPropertyValue("ShadowYDistance", box("int", 150))
```

This routine creates the round edges of both the border shape and the shape that represents the progress in the presentation by adjusting the width of the rectangle. There is also a shadow created for the shapes.

```
68   /* changing the colors of the filling */
69   statusBarShapeProps=statusBarShape~XPropertySet
70   statusBarShapeProps~setPropertyValue("FillStyle",fillStyles~gradient)
71   CALL CreateGradientObject
72   statusBarShapeProps~setPropertyValue("FillGradient", result)
73   statusBarShapeProps~setPropertyValue("LineStyle",-
74     bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
75   CALL setShadowAndFormat(statusBarShapeProps)
```

The shape for the progress bar filling has not just a normal solid color. The *FillStyle* is a *Gradient* Object, which can create a transition from one color to another with detailed configuration how this transition will be done. The *Gradient* Object is generated in an own function. This function returns the object and can be accessed from outside through the variable *result*. Again the function for formatting the shape will be called here.

4.3.2 Example02

```
/* Macro, which generates pacman on his way to his cherries */
/* 02_pacman.rex */
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* initialize all variables (height, width, etc.) */
pagecount=xDrawPages~XIndexAccess~getCount
firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
width = firstDrawPageProps~getPropertyValue("Width")
height = firstDrawPageProps~getPropertyValue("Height")
shapeWidthBorder = width - 1400
shapeHeight = 750
shapeX = 500
shapeY = height - 1500

IF pagecount <= 2 THEN
DO
  .bsf.dialog~messageBox("This presentation has less than three slide. "-
    "There is no need for this macro to run!", "ERROR", "error")
  EXIT 0
END

step = trunc( (width - 2500) / (pagecount - 2))
pacmanPositionX = shapeX

CALL GetPresentationDirectory oDoc~getURL
directory = result
separator = .uno~file.separator

DO i = 1 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage

  /* remove existing pacmans, if necessary */
  xShapes = xDrawPage~XShapes
  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     IF(xShape~XNamed~getName() == "pacman_scene_group") THEN
     DO
        xShapeGroup = xShape~XShapeGroup
        xDrawPage~remove(xShapeGroup)
     END
  END
  /* creating and positioning of pacman */
  pacman = xImpressFactory~createInstance(-
    "com.sun.star.drawing.GraphicObjectShape")
  pacman = pacman~XShape
  CALL setSizeAndPosition pacman, 1000, 1000, pacmanPositionX, shapeY + 120
  pacmanProps=pacman~XPropertySet
  pacmanProps~setPropertyValue("GraphicURL",-
    uno.convertToURL(directory||separator||"pacman.gif"))
  xDrawPage~add(pacman)

  /* creating and positioning of the cherry */
  cherry = xImpressFactory~createInstance(-
    "com.sun.star.drawing.GraphicObjectShape")
  cherry = cherry~XShape
  CALL setSizeAndPosition cherry, 1200, 1200, shapeWidthBorder, shapeY
  cherryProps=cherry~XPropertySet
  cherryProps~setPropertyValue("GraphicURL",-
```

```
67      uno.convertToURL(directory||separator||"cherry.gif"))
68    xDrawPage~add(cherry)
69
70    /* create the group */
71    shapeGroup = xMcf~createInstanceWithContext(-
72      "com.sun.star.drawing.ShapeCollection", xContext)
73    shapeGroup = shapeGroup~XShapes
74    shapeGroup~add(cherry)
75    shapeGroup~add(pacman)
76
77     /* creating and positioning of points */
78    IF (i <> pagecount -1) THEN
79      DO
80        j = pacmanPositionX + 1200
81        DO WHILE j < shapeWidthBorder - 300
82          point = xImpressFactory~createInstance(-
83            "com.sun.star.drawing.EllipseShape")
84          point = point~XShape
85          CALL setSizeAndPosition point, 300, 300, j, shapeY + 500
86          j = j + 800
87          pointProps=point~XPropertySet
88          constant = bsf.getConstant("com.sun.star.drawing.CircleKind", "FULL")
89          pointProps~setPropertyValue("CircleKind", constant)
90          pointProps~setPropertyValue("FillColor", box("int", "FFFF00"x ~c2d))
91          pointProps~setPropertyValue("LineStyle",-
92           bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
93          xDrawPage~add(point)
94          shapeGroup~add(point)
95        END
96      END
97    xShapeGrouper = xDrawPage~XShapeGrouper
98    xShapeGroup = xShapeGrouper~group(shapeGroup)
99    name = xShapeGroup~XNamed
100   name~setName("pacman_scene_group")
101
102   pacmanPositionX = pacmanPositionX + step
103 END
104
105 EXIT 0
106 /* routine for getting the directory of the presentation */
107 GetPresentationDirectory :
108   use arg  path
109
110   separator = .uno~file.separator
111   full = REVERSE(uno.convertFromURL(path))
112   parse var full "pdo." filename (separator) directory
113   directory = REVERSE(directory)
114
115 return directory
116
117 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
118
119 /* routine for positioning and resizing a shape */
120 ::routine  setSizeAndPosition
121   use arg shape, width, height, posX, posY
122
123   shape~setSize(-
124     .bsf~new("com.sun.star.awt.Size", width, height))
125   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
126
127 /* routine for removing selection*/
128 ::routine  removeSelection
129   use arg oDoc
130
131   model= oDoc~XModel
132   controller = model~getCurrentController()
133   selectionController = controller~XSelectionSupplier
```

```
134    selected = selectionController~getSelection()
135    selectionController~select(.nil)
```

Snippet 13: Pacman (02_pacman.rex)

Example number 2 is also a macro, which displays the current status of the presentation. The animation starts on the second slide, like in the first example. On the left side of this slide there will be an Pacman image and on the other side of the slide, there are some cherries. On the line from Pacman to the cherries are a lot of yellow points. Each slide the Pacman image will be shifted to the left. After running the script, it looks like Pacman eats all the points till he finally reaches the cherry on the last slide of the presentation.

Figure 16: Pacman on his way to the cherries

This snippet makes use of another *Shape*, the *GraphicObjectShape*, which loads an image from the hard drive to display it on the slides.

```
50    /* creating and positioning of pacman */
51    pacman = xImpressFactory~createInstance(-
52      "com.sun.star.drawing.GraphicObjectShape")
53    pacman = pacman~XShape
54    CALL setSizeAndPosition pacman, 1000, 1000, pacmanPositionX, shapeY + 120
55    pacmanProps=pacman~XPropertySet
56    pacmanProps~setPropertyValue("GraphicURL",-
57      uno.convertToURL(directory||separator||"pacman.gif"))
58    xDrawPage~add(pacman)
```

The *GraphicObjectShape* is created through the Impress *MultiServiceFactory* interface and is then adjusted to the position and size on the slide. The file URL can be edited by changing the corresponding property value.

```
 77    /* creating and positioning of points */
 78   IF (i <> pagecount -1) THEN
 79     DO
 80       j = pacmanPositionX + 1200
 81       DO WHILE j < shapeWidthBorder - 300
 82         point = xImpressFactory~createInstance(-
 83           "com.sun.star.drawing.EllipseShape")
 84         point = point~XShape
 85         CALL setSizeAndPosition point, 300, 300, j, shapeY + 500
 86         j = j + 800
 87         pointProps=point~XPropertySet
 88         constant = bsf.getConstant("com.sun.star.drawing.CircleKind", "FULL")
 89         pointProps~setPropertyValue("CircleKind", constant)
 90         pointProps~setPropertyValue("FillColor", box("int", "FFFF00"x ~c2d))
 91         pointProps~setPropertyValue("LineStyle",-
 92          bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
 93         xDrawPage~add(point)
 94         shapeGroup~add(point)
 95       END
 96     END
 97   xShapeGrouper = xDrawPage~XShapeGrouper
 98   xShapeGroup = xShapeGrouper~group(shapeGroup)
 99   name = xShapeGroup~XNamed
100   name~setName("pacman_scene_group")
101
102   pacmanPositionX = pacmanPositionX + step
103 END
```

This snippet shows the placing of the circle shapes, which represent the points in the Pacman game. The placement starts at the current position of Pacman and adds *EllipseShape* objects to the drawpage till the x-coordinate reaches the end in form of the position of the cherry.

The circles are just *EllipseShape* objects that have the *CircleKind FULL*. The other options are *SECTION* for a circle with a cut connected by a line, *CUT* for a circle with a cut connected by two lines (for example a cake diagram) and an *ARC*, which is a circle with an open cut.

After the creation of the shapes, they will be added to a *ShapeGroup*, all of the shapes which are inside the *shapeGroup* object will be grouped together using the *XShapeGrouper* from a drawpage. To be able to remove them in the case of a second run of the macro the group gets a name.

4.3.3 Example03

```
/* Macro, which generates a fuse with a bomb, when the fuse has burned down,
   the bomb will explode */
/* 03_bomb.rex */
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* remove the explosion page, if it exists */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount <= 2 THEN
DO
  .bsf.dialog~messageBox("This presentation has less than three slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

DO i = 1 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
  IF(xDrawPage~XNamed~getName() == "explosion_page") THEN
  DO
    xDrawPages~remove(xDrawPage)
    ITERATE
  END
END
/* initialize all variables (height, width, etc.) */
pagecount=xDrawPages~XIndexAccess~getCount
firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
width = firstDrawPageProps~getPropertyValue("Width")
height = firstDrawPageProps~getPropertyValue("Height")
bombPositionX = width - 2500
shapeX = 500
shapeY = height - 2450

step = trunc( (width - 3150) / (pagecount - 2))

positionFlame = shapeX
lengthCord = bombPositionX - 800

CALL GetPresentationDirectory oDoc~getURL
directory = result
separator = .uno~file.separator
DO i = 1 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage

  /* remove existing bombs, if necessary */
  xShapes = xDrawPage~XShapes
  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     IF(xShape~XNamed~getName() == "bomb_group") THEN
     DO
        xShapeGroup = xShape~XShapeGroup
        xDrawPage~remove(xShapeGroup)
     END
  END

  /* creating and positioning of the bomb */
  bomb = xImpressFactory~createInstance(-
    "com.sun.star.drawing.GraphicObjectShape")
  bomb = bomb~XShape
```

```
 67   CALL setSizeAndPosition bomb, 2100, 2150, bombPositionX, shapeY
 68   bombProps=bomb~XPropertySet
 69   bombProps~setPropertyValue("GraphicURL",-
 70         uno.convertToURL(directory||separator||"bomb.gif"))
 71   xDrawPage~add(bomb)
 72
 73    /* creating and positioning of the cord */
 74    cord = xImpressFactory~createInstance(-
 75      "com.sun.star.drawing.RectangleShape")
 76    cord = cord~XShape
 77    CALL setSizeAndPosition cord, lengthCord, 100,-
 78      positionFlame + 700, shapeY + 1300
 79    cordProps=cord~XPropertySet
 80    cordProps~setPropertyValue("FillColor", box("int", "FFFF00"x ~c2d))
 81    xDrawPage~add(cord)
 82
 83    /* creating and positioning of the fuse */
 84    fuse = xImpressFactory~createInstance(-
 85      "com.sun.star.drawing.GraphicObjectShape")
 86    fuse = fuse~XShape
 87    CALL setSizeAndPosition fuse, 1000, 1000, positionFlame, shapeY + 940
 88    fuseProps=fuse~XPropertySet
 89    fuseProps~setPropertyValue("GraphicURL",-
 90      uno.convertToURL(directory||separator||"fuse.png"))
 91    xDrawPage~add(fuse)
 92
 93    /* create the group */
 94    shapeGroup = xMcf~createInstanceWithContext(-
 95      "com.sun.star.drawing.ShapeCollection", xContext)
 96    shapeGroup = shapeGroup~XShapes
 97    shapeGroup~add(bomb)
 98    shapeGroup~add(cord)
 99    shapeGroup~add(fuse)
100    xShapeGrouper = xDrawPage~XShapeGrouper
101    xShapeGroup = xShapeGrouper~group(shapeGroup)
102    name = xShapeGroup~XNamed
103    name~setName("bomb_group")
104
105    positionFlame = positionFlame + step
106    lengthCord = lengthCord - step
107 END
108
109 /* creating, resizing and positioning of the explosion page and content*/
110 xDrawPages~insertNewByIndex(pagecount)
111 explosionPage = xDrawPages~getByIndex(pagecount)~XDrawPage
112 explosionPage~XNamed~setName("explosion_page")
113 textShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
114 textShape = textShape~XShape
115 CALL setSizeAndPosition textShape, 23000, 3000, 3000, 2000
116 explosionPage~add(textShape)
117
118 explosion = xImpressFactory~createInstance(-
119   "com.sun.star.drawing.GraphicObjectShape")
120 explosion = explosion~XShape
121 CALL setSizeAndPosition explosion, 20000, 15000, 4200, 5000
122 explosionPage~add(explosion)
123
124 explosionProps = explosion~XPropertySet
125 explosionProps~setPropertyValue("GraphicURL",-
126   uno.convertToURL(directory||separator||"explosion.jpg"))
127 textProps = textShape~XPropertySet
128 textProps~setPropertyValue("TextFitToSize",-
129     bsf.getConstant("com.sun.star.drawing.TextFitToSizeType","PROPORTIONAL"))
130 effects = bsf.wrapStaticFields("com.sun.star.presentation.AnimationEffect")
131 speeds = bsf.wrapStaticFields("com.sun.star.presentation.AnimationSpeed")
132 textProps~setPropertyValue("Effect", effects~FADE_FROM_CENTER)
133 textProps~setPropertyValue("Speed", speeds~FAST)
```

```
134 explosionProps~setPropertyValue("Effect", effects~HORIZONTAL_ROTATE)
135 explosionProps~setPropertyValue("Speed", speeds~MEDIUM)
136
137 /* formating the textshapes */
138 textShape~XText~setString("BOOOOOOOMMMMM!!!!!!")
139
140 explosionPageProps = explosionPage~XPropertySet
141 explosionPageProps~setPropertyValue("Effect",-
142   bsf.getConstant("com.sun.star.presentation.FadeEffect", "RANDOM"))
143 explosionPageProps~setPropertyValue("Speed", speeds~MEDIUM)
144
145 EXIT 0
146
147 /* routine for getting the directory of the presentation */
148 GetPresentationDirectory :
149   use arg  path
150
151   separator = .uno~file.separator
152   full = REVERSE(uno.convertFromURL(path))
153   parse var full "pdo." filename (separator) directory
154   directory = REVERSE(directory)
155
156 return directory
157
158 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
159
160 /* routine for positioning and resizing a shape */
161 ::routine  setSizeAndPosition
162   use arg shape, width, height, posX, posY
163
164   shape~setSize(-
165     .bsf~new("com.sun.star.awt.Size", width, height))
166   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
167
168 /* routine for removing selection*/
169 ::routine  removeSelection
170   use arg oDoc
171
172   model= oDoc~XModel
173   controller = model~getCurrentController()
174   selectionController = controller~XSelectionSupplier
175   selected = selectionController~getSelection()
176   selectionController~select(.nil)
```

Snippet 14: Bomb (03_bomb.rex)

The type of the next macro is again a progress showing macro. Instead of Pacman going to the end of the slide, there will be a fuse with a flame. The fuse is connected to a bomb. The fuse will be realized with the help of a *RectangleShape*. As the progress of the presentation is advancing, the fuse is burning down. One of the difference to the last example is that the macro is generating a new slide at the end of the presentation and put some shapes on the drawpage. The new drawpage as well as the shapes on it will have some effects, which are assigned in the script.

The last slide contains a *TextShape* with the text “BOOOMMM!!” and an *GraphicObjectShape* with an explosion image.

```
124 explosionProps = explosion~XPropertySet
125 explosionProps~setPropertyValue("GraphicURL",-
126   uno.convertToURL(directory||separator||"explosion.jpg"))
127 textProps = textShape~XPropertySet
128 textProps~setPropertyValue("TextFitToSize",-
129     bsf.getConstant("com.sun.star.drawing.TextFitToSizeType","PROPORTIONAL"))
130 effects = bsf.wrapStaticFields("com.sun.star.presentation.AnimationEffect")
131 speeds = bsf.wrapStaticFields("com.sun.star.presentation.AnimationSpeed")
132 textProps~setPropertyValue("Effect", effects~FADE_FROM_CENTER)
133 textProps~setPropertyValue("Speed", speeds~FAST)
134 explosionProps~setPropertyValue("Effect", effects~HORIZONTAL_ROTATE)
135 explosionProps~setPropertyValue("Speed", speeds~MEDIUM)
```

The URL of the image is set in the cutout using the *XPropertySet* interface. The method used for fetching the filename is operating system independent. The images have to be in the same folder as the presentation. The animation effect of the *TextShape* “BOOOMMMM!!!!” will be a fade from the center of the slide and will be conducted in a fast sped. Similar to that, the effect of the image with the explosion should be a little bit slower and with a horizontal rotation.

```
140 explosionPageProps = explosionPage~XPropertySet
141 explosionPageProps~setPropertyValue("Effect",-
142   bsf.getConstant("com.sun.star.presentation.FadeEffect", "RANDOM"))
143 explosionPageProps~setPropertyValue("Speed", speeds~MEDIUM)
```

The counterpart to the *Shape AnimationEffect* object for slide transitions is the *FadeEffect* object. The *RANDOM* value sets a random effect to the slide transition with medium speed. The bomb with the fuse as well as the slide with the explosion image are displayed on the next two figures.

Figure 17: Bomb with fuse

Figure 18: Explosion slide

For sure, this demonstration of a progress illustration and the example with Pacman is not very appropriate for business presentations. For these kind of presentations the progress bar will be the best solution. This should be just a proof of concept how easy it is to put a very helpful feature to a presentation, also with a whiff of playfulness..

4.3.4 Example04

```
/* Macro, which generates a clock that counts down to the end of the
   presentation */
/* 04_clock.rex */
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* initialize all variables (height, width, etc.) */
pagecount=xDrawPages~XIndexAccess~getCount
firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
width = firstDrawPageProps~getPropertyValue("Width")
height = firstDrawPageProps~getPropertyValue("Height")
bombPositionX = width - 2000
shapeX = 500
shapeY = height - 1250

IF pagecount <= 2 THEN
DO
  .bsf.dialog~messageBox("This presentation has less than three slide! "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

step = trunc((36000) / (pagecount - 2))

/* start at 12 o clock */
endAngle = 9000
startAngle = 9000
stopAngle = -27000
/* start with white */
colorValue = 16777215
stepRed = trunc( 255 / (pagecount - 2))

circleKinds = bsf.wrapStaticFields("com.sun.star.drawing.CircleKind")
DO i = 0 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage

  /* remove existing clocks, if necessary */
  xShapes = xDrawPage~XShapes
  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     IF(xShape~XNamed~getName() == "clock_group") THEN
     DO
        xShapeGroup = xShape~XShapeGroup
        xDrawPage~remove(xShapeGroup)
     END
  END
  IF(i == 0) THEN
    ITERATE
  /* creating background shape of the clock */
  clockBackground = xImpressFactory~createInstance(-
    "com.sun.star.drawing.EllipseShape")
  clockBackground = clockBackground~XShape
  CALL setSizeAndPosition clockBackground, 1500, 1500,-
    trunc(width / 2) - 750, height - 1800
  clockBackgroundProps=clockBackground~XPropertySet
  clockBackgroundProps~setPropertyValue("CircleKind", circleKinds~FULL)
  clockBackgroundProps~setPropertyValue("FillColor",-
    box("int", "FFFFFF"x ~c2d))
  xDrawPage~add(clockBackground)
```

```
67
68    clock = xImpressFactory~createInstance(-
69      "com.sun.star.drawing.EllipseShape")
70    clock = clock~XShape
71    CALL setSizeAndPosition clock, 1500, 1500,-
72      trunc(width / 2) - 750, height - 1800
73    xDrawPage~add(clock)
74    clockProps=clock~XPropertySet
75
76    /* create the group */
77    shapeGroup = xMcf~createInstanceWithContext(-
78      "com.sun.star.drawing.ShapeCollection", xContext)
79    shapeGroup = shapeGroup~XShapes
80    shapeGroup~add(clockBackground)
81    shapeGroup~add(clock)
82    xShapeGrouper = xDrawPage~XShapeGrouper
83    xShapeGroup = xShapeGrouper~group(shapeGroup)
84    name = xShapeGroup~XNamed
85    name~setName("clock_group")
86
87    IF(startAngle <> stopAngle) THEN
88      DO
89        /* set the start and the end angle for the section */
90        clockProps~setPropertyValue("CircleKind", circleKinds~SECTION)
91        clockProps~setPropertyValue("CircleStartAngle", box("int", startAngle))
92        clockProps~setPropertyValue("CircleEndAngle", box("int", endAngle))
93        startAngle = startAngle - step
94        /* adjust color */
95        color = box("int", colorValue)
96        colorValue = colorValue - (stepRed * 256) - stepRed
97      END
98    ELSE
99        color = box("int", "FF0000"x ~c2d)
100
101   clockProps~setPropertyValue("FillColor", color)
102 END
103
104 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
105
106 /* routine for positioning and resizing a shape */
107 ::routine  setSizeAndPosition
108   use arg shape, width, height, posX, posY
109
110   shape~setSize(-
111     .bsf~new("com.sun.star.awt.Size", width, height))
112   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
113
114 /* routine for removing selection*/
115 ::routine  removeSelection
116   use arg oDoc
117
118   model= oDoc~XModel
119   controller = model~getCurrentController()
120   selectionController = controller~XSelectionSupplier
121   selected = selectionController~getSelection()
122   selectionController~select(.nil)
```

Snippet 15: Clock (04_clock.rex)

For this macro the progress is illustrated as a clock on each slide, which turns more and more red as the presentation goes by.

This macro will generate a *EllipseShape* in form of a circle using the property value *CircleKind FULL* for the background of the clock. The shape for displaying the elapsed

time will be a *SECTION EllipseShape*. To configure the section, the values for the start and end angle need to be set.

```
32 /* start at 12 o clock */
33 endAngle = 9000
34 startAngle = 9000

89        /* set the start and the end angle for the section */
90        clockProps~setPropertyValue("CircleKind", circleKinds~SECTION)
91        clockProps~setPropertyValue("CircleStartAngle", box("int", startAngle))
92        clockProps~setPropertyValue("CircleEndAngle", box("int", endAngle))
93        startAngle = startAngle - step
```

By setting the values as seen in the cutout above, the section is forced to start at 12 o'clock. Decreasing the *startAngle* value after each slide will expand the section clockwise. The fill color of the section is also changing throughout the presentation.

```
95        color = box("int", colorValue)
96        colorValue = colorValue - (stepRed * 256) - stepRed
```

The color is represented by a hexadecimal value. Starting with white (FFFFFF) and fading to red. This is done by subtracting values from the start color. The transition of the clocks looks like Figure 19.

Figure 19: Clock transition

4.3.5 Example05

```
/* Macro, which generates a break page with a specfic time to display
   the content. The user can control the duration and the breaktext */
/* 05_break.rex */
/* Exceptionhandling */
SIGNAL ON ANY
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages

/* check pagecount */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount <= 2 THEN
DO
  .bsf.dialog~messageBox("This presentation has less than three slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

/* adjust middle value */
middle = trunc(pagecount / 2);

IF (pagecount // 2 == 0) THEN
   middle = middle - 1

/* ask user what text should be shown and the duration of the break */
minutes = .bsf.dialog~inputBox("How long (minutes) should be the break?",-
  "Question", "question")

/* No input */
if(minutes == "") then
  DO
    .bsf.dialog~messageBox("Not a valid number!", "ERROR", "error")
    EXIT 0
  END
message = .bsf.dialog~inputBox("What text should be displayed?",-
  "Question", "question")

/* insert a slide in the middle of the page */
xDrawPages~insertNewByIndex(middle)

beforeBreakPage = xDrawPages~getByIndex(middle)~XDrawPage
beforeBreakPageProps = beforeBreakPage~XPropertySet
beforeBreakPageProps~setPropertyValue("Effect", -
  bsf.getConstant("com.sun.star.presentation.FadeEffect","WAVYLINE_FROM_LEFT"))

/* set the duration to the entered value */
breakPage = xDrawPages~getByIndex(middle+1)~XDrawPage
breakPageProps = breakPage~XPropertySet
breakPageProps~setPropertyValue("Change", box("int", 1))
breakPageProps~setPropertyValue("Duration", box("int", minutes * 60))

/* create the heading of the break slide */
textShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
textShape = textShape~XShape
CALL setSizeAndPosition textShape, 15000, 3000, 6800, 2000
textProps = textShape~XPropertySet
breakPage~add(textShape)

/* assigning effects */
textProps~setPropertyValue("TextFitToSize",-
    bsf.getConstant("com.sun.star.drawing.TextFitToSizeType","PROPORTIONAL"))
animationEffects = bsf.wrapStaticFields(-
  "com.sun.star.presentation.AnimationEffect")
```

```
 68 speeds = bsf.wrapStaticFields("com.sun.star.presentation.AnimationSpeed")
 69 textProps~setPropertyValue("Effect",animationEffects~RANDOM)
 70 textProps~setPropertyValue("Speed", speeds~MEDIUM)
 71 textShape~XText~setString("-- Break --")
 72
 73 messageShape = xImpressFactory~createInstance(-
 74   "com.sun.star.drawing.TextShape")
 75 messageShape = messageShape~XShape
 76 CALL setSizeAndPosition messageShape, 23000, 2000,-
 77   2800, 7000
 78 messageShapeProps=messageShape~XPropertySet
 79 breakPage~add(messageShape)
 80
 81 /* assigning effects */
 82 messageShapeProps~setPropertyValue("Effect",-
 83   animationEffects~COUNTERCLOCKWISE)
 84 messageShapeProps~setPropertyValue("Speed", speeds~SLOW)
 85 messageShapeProps~setPropertyValue("FillStyle",-
 86     bsf.getConstant("com.sun.star.drawing.FillStyle", "NONE"))
 87 messageShapeProps~setPropertyValue("LineStyle",-
 88   bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
 89
 90 breakText = messageShape~XText
 91 breakText~setString(message)
 92
 93 EXIT 0
 94 ANY:
 95    .bsf.dialog~messageBox("Error in line " SIGL "."-
 96      "Please check your input!", "ERROR", "error")
 97 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
 98
 99 /* routine for positioning and resizing a shape */
100 ::routine  setSizeAndPosition
101   use arg shape, width, height, posX, posY
102
103   shape~setSize(-
104     .bsf~new("com.sun.star.awt.Size", width, height))
105   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
```

Snippet 16: Create Break Slide (05_break.rex)

The next macro will support the presenter by automatically creating a slide in the middle of the presentation. For long presentations it could be necessary to make such a break to let the audience get some refreshments. The user will be asked for the duration of the break and for a message, which will be shown on the new slide. The duration has to be entered in minutes. After the time is elapsed the presentation will continue with the next slide.

The slide before the break as well as all created shapes will get effects in addition to the slide transition.

```
5 SIGNAL ON ANY
```

This line is responsible for the exception handling in Open Object Rexx. If any error occurs during the runtime of the script, the code at the marker *ANY* will be executed. The script has to handle possible exceptions because it has to operate with a data entered by a user. When the user does not enter a valid number, there would be a SYNTAX error and the script would crash.

In order to handle those problems, the marker *ANY* was implemented in the rear section of the script.

```
94 ANY:
95     .bsf.dialog~messageBox("Error in line " SIGL "."-
96       "Please check your input!", "ERROR", "error")
97 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
```

The code then shows up another message box with the line number where the problem occurred.

```
23 /* adjust middle value */
24 middle = trunc(pagecount / 2);
25
26 IF (pagecount // 2 == 0) THEN
27    middle = middle - 1
28
29 /* ask user what text should be shown and the duration of the break */
30 minutes = .bsf.dialog~inputBox("How long (minutes) should be the break?",-
31   "Question", "question")
```

The cutout from the snippet above shows how the middle of the presentation is been calculated. The // operator acts as the modulo function in Open Object Rexx to get the rest of a division of integers.
The *inputBox* procedure from the *BSF.CLS* module is used for entering the duration of the break. The value will be saved into the variable *minutes*.

The dialogs and the break slide can be seen on the next figures.

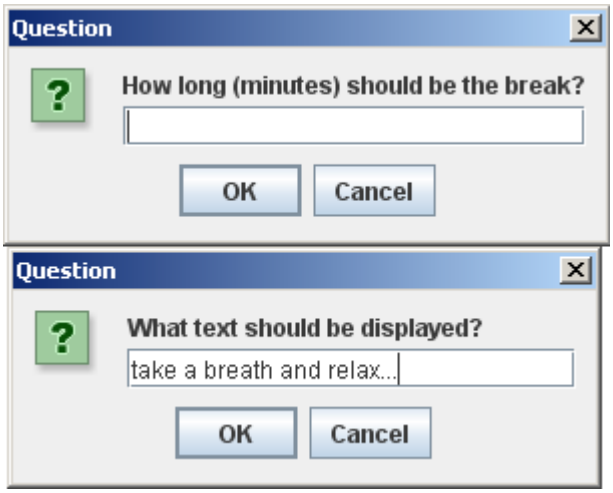

Figure 20: Input dialogs

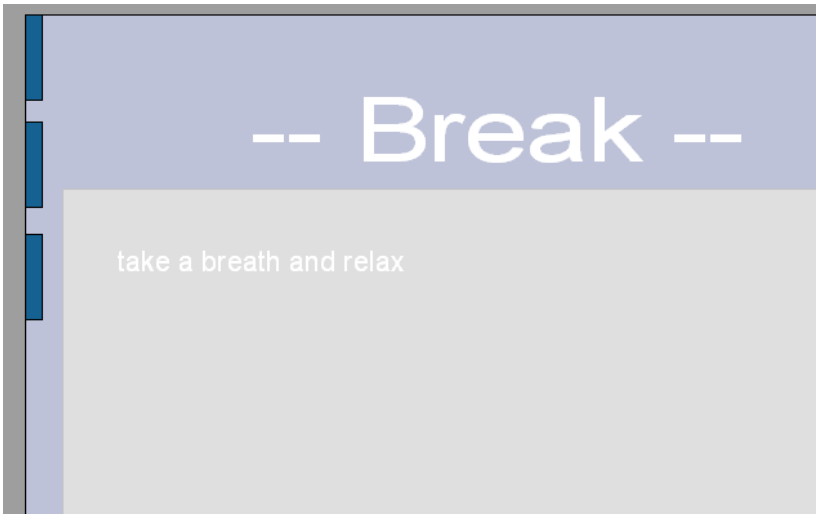

Figure 21: Generated break slide

4.3.6 Example06

```
/* Macro, which generates different kinds of animations and effects
   the user will be asked for the settings of the presentation */
/* 06_finish_presentation.rex */
/* Exceptionhandling */
SIGNAL ON SYNTAX
xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages

/* check pagecount */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount == 1 THEN
DO
  .bsf.dialog~messageBox("This presentation has only one slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

/* show message dialog for slide effects */
arrayEffects = .array~of("NONE", "RANDOM", "DISSOLVE", "CLOCKWISE", -
 "MOVE_FROM_TOP", "FADE_FROM_BOTTOM", "HORIZONTAL_STRIPES", -
 "SPIRALIN_RIGHT", "CLOSE_HORIZONTAL", "ROLL_FROM_TOP")
effect = .bsf.dialog~inputBox("Choose the Effect you like to add to "-
  "the slides!", "Effects", "question", , arrayEffects)
IF(effect == .nil) THEN
  EXIT 0

/* show message dialog for speed of transition */
IF(effect <> "NONE") THEN
DO
  arraySpeed = .array~of("SLOW", "MEDIUM", "FAST")
  speed = .bsf.dialog~inputBox("Choose the speed of the transition!",-
    "Effects", "question", , arraySpeed)
  IF(speed == .nil) THEN
    EXIT 0
END

/* ask for the duration of the transition */
linebreak = "0d0a"x
DO UNTIL seconds <> ""
  seconds = .bsf.dialog~inputBox("How long (seconds) should be the ",
    "duration of a slide"||linebreak||"0= manual transition?",-
    "Question", "question")
  if(seconds == .nil) then
    EXIT 0
  /* No input */
  if(seconds == "") then
      .bsf.dialog~messageBox("Not a valid number!", "ERROR", "error")
END

shapeAnimation = .bsf.dialog~dialogBox("Do you want a shape animation?"-
  , "Question", "question", "YesNo")

/* ask for the global shape animation */
IF (shapeAnimation == 0) THEN
DO
  arrayEffects = .array~of("NONE", "RANDOM", "DISSOLVE", "APPEAR", -
   "LASER_FROM_LEFT", "MOVE_TO_BOTTOM", "STRETCH_FROM_LEFT", -
   "WAYLINE_FROM_RIGHT", "ZOOM_OUT", "VERTICAL_LINES")
  effectShapes = .bsf.dialog~inputBox("Choose the Effect you like to add to "-
    "the shapes!", "Effects", "question", , arrayEffects)
  IF(effectShapes == .nil) THEN
```

```
 67      EXIT 0
 68
 69    /* ask for the speed of the transition */
 70    IF(effectShapes <> "NONE") THEN
 71    DO
 72      arraySpeed = .array~of("SLOW", "MEDIUM", "FAST")
 73      speedShape = .bsf.dialog~inputBox("Choose the speed of the shape "-
 74        "animation!", "Effects", "question", , arraySpeed)
 75      IF(speedShape == .nil) THEN
 76        EXIT 0
 77    END
 78  END
 79
 80  /* ask for end-slide and autmatic start of the presentation */
 81  addEndSlide = .bsf.dialog~dialogBox("Should a 'Thank you for your attention'"-
 82    "slide be generated?", "Question", "question", "YesNo")
 83
 84  runPresentation = .bsf.dialog~dialogbox("Do you want to start the "-
 85   "presentation after adding the new slide?", "Question", "question", "YesNo")
 86
 87  IF (addEndSlide == 0) THEN
 88    DO
 89      /* insert a slide at the end of the presentation */
 90      xDrawPages~insertNewByIndex(pagecount)
 91      lastPage = xDrawPages~getByIndex(pagecount)~XDrawPage
 92
 93      /* create the heading of the last slide */
 94      textShape = xImpressFactory~createInstance("com.sun.star.drawing.TextShape")
 95      textShape = textShape~XShape
 96      CALL setSizeAndPosition textShape, 22000, 2200, 2800, 7700
 97      textProps = textShape~XPropertySet
 98      lastPage~add(textShape)
 99      textProps~setPropertyValue("TextFitToSize",-
100        bsf.getConstant("com.sun.star.drawing.TextFitToSizeType","PROPORTIONAL"))
101      textShape~XText~setString("Thank you for your attention!")
102    END
103
104  /* go through each slide */
105  DO i = 0 TO pagecount - 1
106    xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
107    xShapes = xDrawPage~XShapes
108  /* go over each shape if wanted */
109    IF(shapeAnimation == 0) THEN
110      DO j = 0 TO xShapes~getCount - 1
111         xShape = xShapes~getByIndex(j)
112         xShapeProps = xShape~XPropertySet
113         /* assigning effects, etc. */
114         xShapeProps~setPropertyValue("Effect",-
115           bsf.getConstant("com.sun.star.presentation.AnimationEffect",-
116           effectShapes))
117         IF(speedShape <> "SPEEDSHAPE") THEN
118         xShapeProps~setPropertyValue("Speed",-
119           bsf.getConstant("com.sun.star.presentation.AnimationSpeed",-
120           speedShape))
121      END
122    xDrawPageProps = xDrawPage~XPropertySet
123    IF (seconds == 0) THEN
124      value = 0
125    ELSE
126      value = 1
127    /* assigning effects, etc. */
128    xDrawPageProps~setPropertyValue("Change", box("int", value))
129    xDrawPageProps~setPropertyValue("Duration", box("int", seconds))
130    xDrawPageProps~setPropertyValue("Effect", -
131      bsf.getConstant("com.sun.star.presentation.FadeEffect", effect))
132    IF(speed <> "SPEED") THEN
133      xDrawPageProps~setPropertyValue("Speed", -
```

```
134       bsf.getConstant("com.sun.star.presentation.AnimationSpeed", speed))
135 END
136
137 /* starting the presentation */
138 if(runPresentation == 0) THEN
139   DO
140     xPresentation = oDoc~XPresentationSupplier~getPresentation
141     xPresentation~bsf.invoke("start")
142   END
143
144 EXIT 0
145 SYNTAX:
146    .bsf.dialog~messageBox("Error in line " SIGL ". Please check your input!",-
147      "ERROR", "error")
148
149 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
150
151 /* routine for positioning and resizing a shape */
152 ::routine  setSizeAndPosition
153   use arg shape, width, height, posX, posY
154
155   shape~setSize(-
156     .bsf~new("com.sun.star.awt.Size", width, height))
157   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
```

Snippet 17: Various tasks (06_finish_presentation.rex)

Macro number 6 carries out some recurring tasks, which can cost a lot of time if one has to do it manually. The user will be asked for each function that should be added to the presentation. The following functions are support by this macro:

- Fade effects and duration of the slide transitions (the user can choose from some predefined effects and speeds)
- Animation effects for every Shape object of the draw pages (again, the user can choose from different effects)
- Automatically generated “Thanks for your attention” slide
- Direct start of the presentation at the end of the macro

```
23 /* show message dialog for slide effects */
24 arrayEffects = .array~of("NONE", "RANDOM", "DISSOLVE", "CLOCKWISE", -
25  "MOVE_FROM_TOP", "FADE_FROM_BOTTOM", "HORIZONTAL_STRIPES", -
26  "SPIRALIN_RIGHT", "CLOSE_HORIZONTAL", "ROLL_FROM_TOP")
27 effect = .bsf.dialog~inputBox("Choose the Effect you like to add to "-
28   "the slides!", "Effects", "question", , arrayEffects)
29 IF(effect == .nil) THEN
30   EXIT 0
```

This cutout shows how arrays can be created in Open Object Rexx using the *.array~of* method. Then the user will be asked in form of a dialog with a combo box, which displays the effects from the array. If the user clicks on “Cancel” in the dialog, the

returned value will be the nil object.

```
43 linebreak = "0d0a"x
```

The hexadecimal value 0D0A represents a line break and is used in a dialog box to separate the message.

```
104 /* go through each slide */
105 DO i = 0 TO pagecount - 1
106   xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
107   xShapes = xDrawPage~XShapes
108 /* go over each shape if wanted */
109   IF(shapeAnimation == 0) THEN
110     DO j = 0 TO xShapes~getCount - 1
111       xShape = xShapes~getByIndex(j)
112       xShapeProps = xShape~XPropertySet
113       /* assigning effects, etc. */
114       xShapeProps~setPropertyValue("Effect",-
115         bsf.getConstant("com.sun.star.presentation.AnimationEffect",-
116         effectShapes))
117       IF(speedShape <> "SPEEDSHAPE") THEN
118       xShapeProps~setPropertyValue("Speed",-
119         bsf.getConstant("com.sun.star.presentation.AnimationSpeed",-
120         speedShape))
```

The macro can also change the animation effects of every shape on every drawpage if the user wants to. Using the interface *XShapes* of a drawpage one can get access to every shape object on it. Like iterating through the drawpages of a presentation, one can access the shapes using an index.

The dialogs for asking after the effects are shown on the next figures.

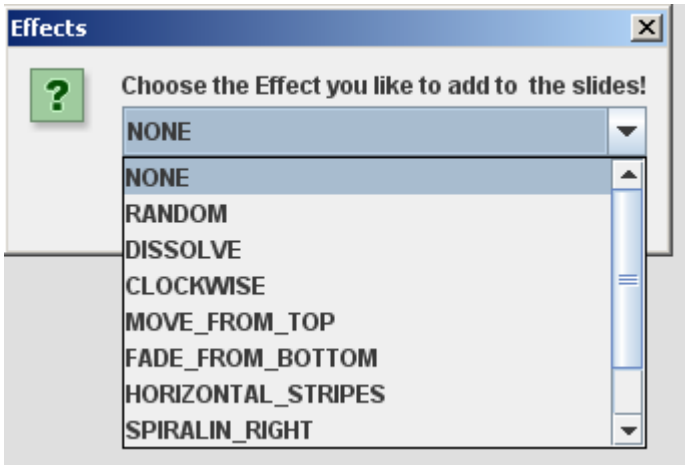

Figure 22: Slide effects

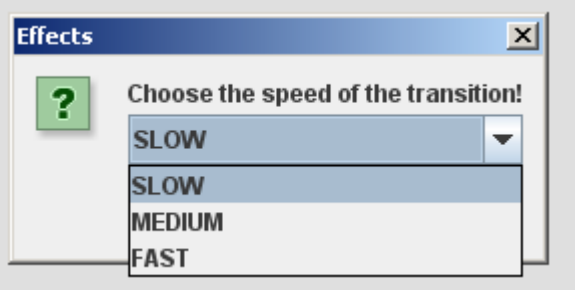

Figure 23: Dialog for choosing the speed

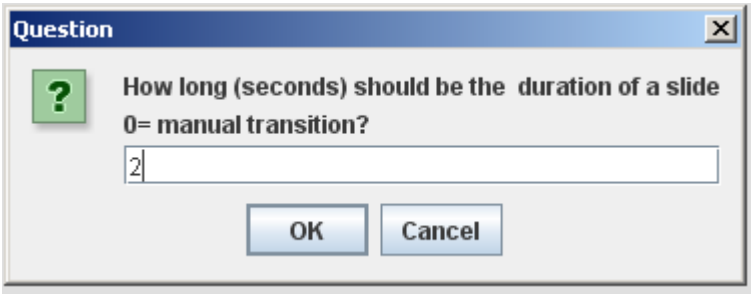

Figure 24: Duration of one slide

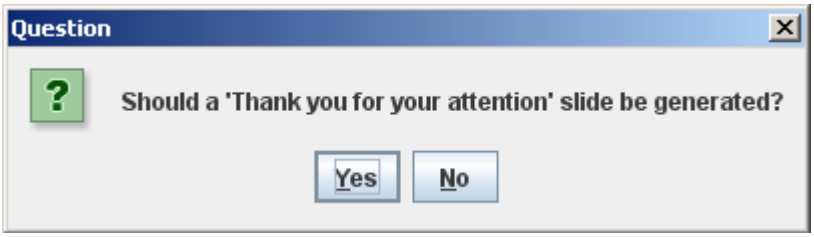

Figure 25: Question dialog for end slide

4.3.7 Example07

```
/* This macro generates a guidepost section on the left side of each
   page. It shows all the heading 1 textshapes on it and marks the
   current position */
/* 07_guideposts.rex */

xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* check pagecount */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount == 1 THEN
DO
  .bsf.dialog~messageBox("This presentation has only one slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
height = firstDrawPageProps~getPropertyValue("Height")

/* ask for end-slide, the slide will get no guideposts */
hasEndSlide = .bsf.dialog~dialogBox("Is there an end-slide in "-
  "this presentation?", "Question", "question", "YesNo")
headlineName = getHeadlineDisplayName(oDoc, "headline")
headline1Name = getHeadlineDisplayName(oDoc, "headline1")
/* create array with index and title of each heading-slide */
headingIndex = .array ~new
counter = 0
startIndex = 0
DO i = 0 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage

  /* remove existing guideposts, if necessary */
  xShapes = xDrawPage~XShapes
  IF(xShapes~getCount > 0) THEN
    DO j = 0 TO xShapes~getCount - 1
       xShape = xShapes~getByIndex(j)
       IF(xShape~XNamed~getName() == "guidepost_group") THEN
       DO
          xShapeGroup = xShape~XShapeGroup
          xDrawPage~remove(xShapeGroup)
       END
    END

  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     xShapeProps = xShape~XPropertySet
     style = xShapeProps~getPropertyValue("Style")
     styleProps = style~XPropertySet
     nameStyle = styleProps~getPropertyValue("DisplayName")
     IF(xShape~XText == .nil) THEN
        ITERATE
     text = xShape~XText~getString()
     /* if the style is heading */
     IF (nameStyle == headlineName) THEN
     DO
        IF (startIndex == 0) THEN
           startIndex = i
```

```
 67          headingIndex~put(i||":"||"1:"||text, counter+1)
 68          counter = counter + 1
 69        END
 70        IF (nameStyle == headline1Name) THEN
 71        DO
 72          IF (startIndex == 0) THEN
 73            startIndex = i
 74          headingIndex~put(i||":"||"2:"||text, counter+1)
 75          counter = counter + 1
 76        END
 77    END
 78  END
 79
 80  /* there are no heading slides */
 81  IF counter == 0 THEN
 82  DO
 83    .bsf.dialog~messageBox("This presentation has no heading textfields. "-
 84      "There is no need for running this macro!", "ERROR", "error")
 85    EXIT 0
 86  END
 87
 88  IF hasEndSlide = 0 THEN
 89    endIndex = pagecount - 2
 90  ELSE
 91    endIndex = pagecount - 1
 92
 93  stepY = trunc((height - 3000) / counter)
 94  posY = 3000
 95  DO i = startIndex TO endIndex
 96    xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
 97      /* creating and positioning of left rectangle with the guideposts */
 98    rectangle = xImpressFactory~createInstance(-
 99      "com.sun.star.drawing.RectangleShape")
100    rectangle = rectangle~XShape
101    CALL setSizeAndPosition rectangle, 6000, height, 50, 50
102    rectangleProps=rectangle~XPropertySet
103    rectangleProps~setPropertyValue("FillColor", box("int", "BABED6"x ~c2d))
104    rectangleProps~setPropertyValue("LineStyle",-
105      bsf.getConstant("com.sun.star.drawing.LineStyle", "NONE"))
106    xDrawPage~add(rectangle)
107
108    /* create the group */
109    shapeGroup = xMcf~createInstanceWithContext(-
110      "com.sun.star.drawing.ShapeCollection", xContext)
111    shapeGroup = shapeGroup~XShapes
112    shapeGroup~add(rectangle)
113
114    posY = 3000
115    counter = 1
116    marked = 0
117    /* adding the headings to the rectangle and mark the correct heading */
118    DO item OVER headingIndex
119      PARSE VAR item id":"level":"textGuidePost
120      nextItem = headingIndex[counter+1]
121      PARSE VAR nextItem nextIndex ":"
122
123      textShape = xImpressFactory~createInstance(-
124        "com.sun.star.drawing.TextShape")
125      textShape = textShape~XShape
126
127      textProps = textShape~XPropertySet
128      xDrawPage~add(textShape)
129      shapeGroup~add(textShape)
130      textShape~XText~setString(textGuidePost)
131      IF(counter <= i & i < nextIndex & marked == 0) THEN
132      DO
133        textProps~setPropertyValue("CharColor",  box("int", "FF0000"x ~c2d))
```

```
134       marked = 1
135     END
136     ELSE
137       textProps~setPropertyValue("CharColor",  box("int", "000000"x ~c2d))
138     counter = counter + 1
139     padding = 0
140     IF(level == 1) THEN
141       textProps~setPropertyValue("CharHeight", box("float", 26))
142     ELSE
143       DO
144         textProps~setPropertyValue("CharHeight", box("float", 20))
145         padding = 600
146       END
147     CALL setSizeAndPosition textShape, 5400, 1200, 300 + padding, posY
148     posY = posY + stepY
149   END
150
151   xShapeGrouper = xDrawPage~XShapeGrouper
152   xShapeGroup = xShapeGrouper~group(shapeGroup)
153   name = xShapeGroup~XNamed
154   name~setName("guidepost_group")
155 END
156 EXIT 0
157
158 getHeadlineDisplayName :
159   oDoc = ARG(1)
160   progName = ARG(2)
161   model= oDoc~XModel
162   famSupplier = model~XStyleFamiliesSupplier
163   families = famSupplier~getStyleFamilies()
164   graphs = families~getByName("graphics")
165   styles = graphs~XNameAccess
166   titelStyle = styles~getByName(progName)
167   styleProps = titelStyle~XPropertySet
168   RETURN styleProps~getPropertyValue("DisplayName")
169
170 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
171
172 /* routine for positioning and resizing a shape */
173 ::routine  setSizeAndPosition
174   use arg shape, width, height, posX, posY
175
176   shape~setSize(-
177     .bsf~new("com.sun.star.awt.Size", width, height))
178   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
179
180 /* routine for removing selection*/
181 ::routine  removeSelection
182   use arg oDoc
183
184   model= oDoc~XModel
185   controller = model~getCurrentController()
186   selectionController = controller~XSelectionSupplier
187   selected = selectionController~getSelection()
188   selectionController~select(.nil)
```

Snippet 18: Guideposts from headings (07_guideposts.rex)

The goal of this macro is to create guideposts on the slides of a presentation. The only thing the creator of the presentation has to care about is that the slides have text fields with the predefined style *Heading* or *Heading1* assigned to them. Then the macro

scans each slide for those heading tagged shapes. After gathering this information, the script creates a rectangular section on the left side of the slides and puts the headings on it. To give the audience the chance to be aware of the status of the presentation, the current heading text is red colored. Figure 26 shows a part of a slide, which was changed by the macro. The macro supports a guidepost structure up to two levels.

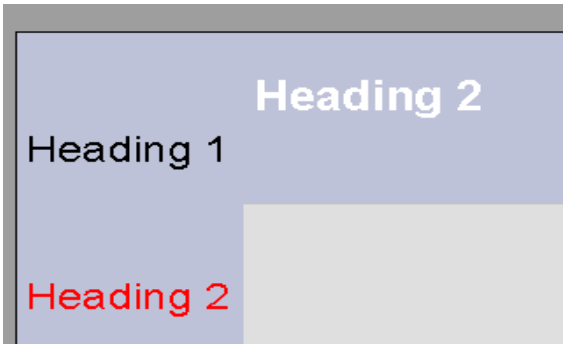

Figure 26: Guideposts from heading

To make sure that there will be no guidepost on the last slide (this slide can be for example a “Thank you for your attention” slide) the user will be asked for that, before the start of the macro's main job.

```
34 /* create array with index and title of each heading-slide */
35 headingIndex = .array ~new
36 counter = 0
37 startIndex = 0
38 DO i = 0 TO pagecount - 1
39   xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
53   DO j = 0 TO xShapes~getCount - 1
54      xShape = xShapes~getByIndex(j)
55      xShapeProps = xShape~XPropertySet
56      style = xShapeProps~getPropertyValue("Style")
57      styleProps = style~XPropertySet
58      nameStyle = styleProps~getPropertyValue("DisplayName")
59      IF(xShape~XText == .nil) THEN
60         ITERATE
61      text = xShape~XText~getString()
62      /* if the style is heading */
63      IF (nameStyle == headlineName) THEN
64      DO
65         IF (startIndex == 0) THEN
66            startIndex = i
67         headingIndex~put(i||":"||"1:"||text, counter+1)
```

The first thing to do is to iterate through all the slides to get every drawpage. With the drawpage, one can access every shape that is added to it. The property *Style* has itself another property called *DisplayName.* If the name is “Heading*”* the index of the relevant slide and the text of the heading is added to the array. There exists also a counter variable to calculate the span between the headings on the guidepost-section. The

missing lines of the snippet are just removing previous guideposts if necessary. The code is the same as in the examples for creating a progress information.

```
117   /* adding the headings to the rectangle and mark the correct heading */
118   DO item OVER headingIndex
119     PARSE VAR item id":"level":"textGuidePost
120     nextItem = headingIndex[counter+1]
121     PARSE VAR nextItem nextIndex ":"
122
123     textShape = xImpressFactory~createInstance(-
124       "com.sun.star.drawing.TextShape")
125     textShape = textShape~XShape
126
127     textProps = textShape~XPropertySet
128     xDrawPage~add(textShape)
129     shapeGroup~add(textShape)
130     textShape~XText~setString(textGuidePost)
131     IF(counter <= i & i < nextIndex & marked == 0) THEN
132     DO
133       textProps~setPropertyValue("CharColor",  box("int", "FF0000"x ~c2d))
134       marked = 1
135     END
136     ELSE
137       textProps~setPropertyValue("CharColor",  box("int", "000000"x ~c2d))
```

The next cutout is responsible for creating the heading fields in the guidepost-section of the slides. In a *DO OVER* iteration all the heading fields in the array are examined with the help of the *PARSE VAR* command. This command puts the slide index and the text of the heading into different variables. As seen in line 66 of the previous cutout, this information is stored as one literal into the array.

Depending on the current index of the slide and the index of the next slide, the heading will be colored in red or in black.

4.3.8 Example08

```
/* This macro does exactly the same as 07_guideposts, but displays the
   headings in form of circles on the slides and marks it */
/* 08_guideposts_circles.rex */

xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages
/* global service manager for shape grouper */
xContext = xScriptContext~getComponentContext
XMcf = xContext~getServiceManager
CALL removeSelection oDoc
/* check pagecount */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount == 1 THEN
DO
  .bsf.dialog~messageBox("This presentation has only one slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

firstDrawPageProps = xDrawPages~getByIndex(0)~XDrawPage~XPropertySet
height = firstDrawPageProps~getPropertyValue("Height")

/* ask for end-slide, the slide will get no guideposts */
hasEndSlide = .bsf.dialog~dialogBox("Is there an end-slide in "-
  "this presentation?", "Question", "question", "YesNo")
headlineName = getHeadlineDisplayName(oDoc)
/* create array with index and title of each heading-slide */
headingIndex = .array ~new
counter = 0
startIndex = 0
DO i = 0 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
  /* remove existing guideposts, if necessary */
  xShapes = xDrawPage~XShapes
  IF(xShapes~getCount > 0) THEN
    DO j = 0 TO xShapes~getCount - 1
       xShape = xShapes~getByIndex(j)
       IF(xShape~XNamed~getName() == "guidepost_group") THEN
       DO
          xShapeGroup = xShape~XShapeGroup
          xDrawPage~remove(xShapeGroup)
       END
    END

  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     xShapeProps = xShape~XPropertySet
     style = xShapeProps~getPropertyValue("Style")
     styleProps = style~XPropertySet
     nameStyle = styleProps~getPropertyValue("DisplayName")
     IF(xShape~XText == .nil) THEN
       ITERATE
     text = xShape~XText~getString()
/* if the style is heading */
     IF (nameStyle == headlineName) THEN
     DO
        IF (startIndex == 0) THEN
          startIndex = i
/* assign bookmark to each heading page */
        xPageName = xDrawPage~XNamed
        xPageName~setName(i)
```

```
 67          headingIndex~put(i, counter+1)
 68          counter = counter + 1
 69        END
 70    END
 71  END
 72
 73  /* there are no heading slides */
 74  IF counter == 0 THEN
 75  DO
 76    .bsf.dialog~messageBox("This presentation has no heading textfields. "-
 77      "There is no need for running this macro!", "ERROR", "error")
 78    EXIT 0
 79  END
 80
 81  IF hasEndSlide = 0 THEN
 82    endIndex = pagecount - 2
 83  ELSE
 84    endIndex = pagecount - 1
 85
 86  stepY = trunc((height - 3000) / counter)
 87  posY = 3000
 88  countHeadings = counter
 89  DO i = startIndex TO endIndex
 90    xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
 91    posY = 3000
 92    counter = 1
 93    marked = 0
 94    /* create the group */
 95    shapeGroup = xMcf~createInstanceWithContext(-
 96      "com.sun.star.drawing.ShapeCollection", xContext)
 97    shapeGroup = shapeGroup~XShapes
 98
 99    DO itemId OVER headingIndex
100      nextIndex = headingIndex[counter+1]
101      IF (countHeadings <> counter) THEN
102      DO
103  /* adding the lines between the heading circles till the last guidepost */
104        rectangle = xImpressFactory~createInstance(-
105          "com.sun.star.drawing.RectangleShape")
106        rectangle = rectangle~XShape
107        CALL setSizeAndPosition rectangle, 300, stepY - 1000, 1360, posY + 1300
108        rectangleProps=rectangle~XPropertySet
109        IF (nextIndex <= i) THEN
110          rectangleProps~setPropertyValue("FillColor", box("int", "FF0000"x ~c2d))
111        ELSE
112          rectangleProps~setPropertyValue("FillColor",-
113            box("int", "000000"x ~c2d))
114        xDrawPage~add(rectangle)
115        shapeGroup~add(rectangle)
116      END
117
118  /* create the circles with the heading serial number */
119      point = xImpressFactory~createInstance(-
120        "com.sun.star.drawing.EllipseShape")
121      point = point~XShape
122      pointProps=point~XPropertySet
123      constant = bsf.getConstant("com.sun.star.drawing.CircleKind", "FULL")
124      pointProps~setPropertyValue("CircleKind", constant)
125      pointProps~setPropertyValue("FillColor", box("int", "FFFF00"x ~c2d))
126      CALL setSizeAndPosition point, 1400, 1400, 800, posY
127      xDrawPage~add(point)
128      shapeGroup~add(point)
129
130  /* set the click event to the bookmark of the corresponding heading */
131      pointProps~setPropertyValue("OnClick", -
132        bsf.getConstant("com.sun.star.presentation.ClickAction","BOOKMARK"))
133      pointProps~setPropertyValue("Bookmark", itemId)
```

```
134     point~XText~setString(counter)
135     posY = posY + stepY
136
137     IF(counter <= i & i < nextIndex & marked == 0) THEN
138     DO
139       pointProps~setPropertyValue("CharColor",  box("int", "FF0000"x ~c2d))
140       marked = 1
141     END
142     ELSE
143       pointProps~setPropertyValue("CharColor",  box("int", "000000"x ~c2d))
144     counter = counter + 1
145   END
146   xShapeGrouper = xDrawPage~XShapeGrouper
147   xShapeGroup = xShapeGrouper~group(shapeGroup)
148   name = xShapeGroup~XNamed
149   name~setName("guidepost_group")
150 END
151 EXIT 0
152
153 getHeadlineDisplayName :
154   oDoc = ARG(1)
155   model= oDoc~XModel
156   famSupplier = model~XStyleFamiliesSupplier
157   families = famSupplier~getStyleFamilies()
158   graphs = families~getByName("graphics")
159   styles = graphs~XNameAccess
160   titelStyle = styles~getByName("headline")
161   styleProps = titelStyle~XPropertySet
162   RETURN styleProps~getPropertyValue("DisplayName")
163
164 ::requires UNO.CLS    -- load UNO support for OpenOffice.org
165
166 /* routine for positioning and resizing a shape */
167 ::routine  setSizeAndPosition
168   use arg shape, width, height, posX, posY
169
170   shape~setSize(-
171     .bsf~new("com.sun.star.awt.Size", width, height))
172   shape~setPosition(.bsf~new("com.sun.star.awt.Point", posX, posY))
173
174 /* routine for removing selection*/
175 ::routine  removeSelection
176   use arg oDoc
177
178   model= oDoc~XModel
179   controller = model~getCurrentController()
180   selectionController = controller~XSelectionSupplier
181   selected = selectionController~getSelection()
182   selectionController~select(.nil)
```

Snippet 19: Guideposts with circles and bookmarks (08_guideposts_circles.rex)

Macro number 8 is also adding a guidepost-section to the slides of a presentation. There are two differences compared to the macro number 7 above. From the graphical point of view, the guideposts will be displayed as circles connected by lines. Instead of the text of the heading a serial number is shown on the slides. The color of the numbers as well as the lines, which connect them, will be changed as the presentation goes by. The result can be seen on Figure 27.

But the most important difference is the use of bookmarks in this script. Each guidepost gets an *OnClick* event, which directly leads to the slide with the corresponding heading. This will provide a fast and easy way of navigating through the presentation without writing those actions and graphical shapes manually.

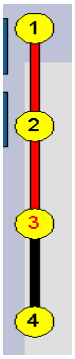

Figure 27: Guideposts with bookmarks

Only the parts of the code, which are different compared to Snippet 18 will be shown in this paragraph.

```
64 /* assign bookmark to each heading page */
65          xPageName = xDrawPage~XNamed
66          xPageName~setName(i)
```

When the macro has found a slide with a heading text field, the related drawpage gets a name, to be able to access it later with a bookmark. The name is set with the help of the *XNamed* interface and is the index of the heading slide.

These lines set the name of the drawpage to the index of the slide.

```
130 /* set the click event to the bookmark of the corresponding heading */
131     pointProps~setPropertyValue("OnClick", -
132       bsf.getConstant("com.sun.star.presentation.ClickAction","BOOKMARK"))
133     pointProps~setPropertyValue("Bookmark", itemId)
134     point~XText~setString(counter)
```

This cutout shows how the *OnClick* event is been connected to the bookmark using the index of the guidepost circle shape. The text inside the circle will be a serial number and is assigned to the shape in the line 134.

4.3.9 Example09

```
/* This macro creates automatically an agenda of the presentation
   using the heading1 textfields */
/* 09_agenda.rex */

xScriptContext=uno.getScriptContext()  -- get the xScriptContext object
oDoc=xScriptContext~getDocument  -- get the document service (an XModel object)
/* retrieving the important interfaces to get access to the drawpages */
xDrawPagesSupplier=oDoc~XDrawPagesSupplier
xImpressFactory = oDoc~XMultiServiceFactory
xDrawPages = xDrawPagesSupplier~getDrawPages

/* check pagecount */
pagecount=xDrawPages~XIndexAccess~getCount

IF pagecount == 1 THEN
DO
  .bsf.dialog~messageBox("This presentation has only one slide. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END
headlineName = getHeadlineDisplayName(oDoc)
/* create array with index and title of each heading-slide */
headingIndex = .array ~new
counter = 0
startIndex = 0
DO i = 0 TO pagecount - 1
  xDrawPage = xDrawPages~getByIndex(i)~XDrawPage
  xShapes = xDrawPage~XShapes
  DO j = 0 TO xShapes~getCount - 1
     xShape = xShapes~getByIndex(j)
     xShapeProps = xShape~XPropertySet
     style = xShapeProps~getPropertyValue("Style")
     styleProps = style~XPropertySet
     nameStyle = styleProps~getPropertyValue("DisplayName")
     IF(xShape~XText == .nil) THEN
       ITERATE
     text = xShape~XText~getString()
     /* if the style is heading */
     IF (nameStyle == headlineName) THEN
     DO
        IF (startIndex == 0) THEN
           startIndex = i
        headingIndex~put(i||":"||text, counter+1)
        counter = counter + 1
     END
  END
END

/* there are no heading slides */
IF counter == 0 THEN
DO
  .bsf.dialog~messageBox("This presentation has no heading textfields. "-
    "There is no need for running this macro!", "ERROR", "error")
  EXIT 0
END

/* if there is no existing agenda insert a new drawpage */
secondSlide = xDrawPages~getByIndex(1)~XDrawPage
nameSecondSlide = secondSlide~XNamed
IF (nameSecondSlide~getName() <> "agenda") THEN
DO
  xDrawPages~insertNewByIndex(0)
END

/* format the agenda slide */
agendaPage = xDrawPages~getByIndex(1)~XDrawPage
```

```
 67 CALL prepareAgendaSlide agendaPage
 68
 69 /* positioning of the text cursor */
 70 headingListShape = agendaPage~XShapes
 71 headingListShape = headingListShape~getByIndex(1)
 72 xText = headingListShape~XText
 73 xText~setString("")
 74 xTextCursor = xText~createTextCursor
 75 xTextCursor~gotoEnd(.false)
 76 xTextRange = xTextCursor~XTextRange
 77
 78 linebreak = "0d0a"x
 79 start = 0
 80
 81 DO item OVER headingIndex
 82   PARSE VAR item id":"text
 83   /* adding the headings to the listing */
 84   IF (start == 1) THEN
 85     xTextRange~setString(linebreak||text)
 86   ELSE
 87   DO
 88     xTextRange~setString(text)
 89     start = 1
 90   END
 91   xTextCursor~gotoEnd(.false)
 92   xTextRange = xTextCursor~XTextRange
 93 END
 94
 95 /* format the listing */
 96 CALL formatAgenda headingListShape
 97 EXIT 0
 98
 99 getHeadlineDisplayName :
100   oDoc = ARG(1)
101   model= oDoc~XModel
102   famSupplier = model~XStyleFamiliesSupplier
103   families = famSupplier~getStyleFamilies()
104   graphs = families~getByName("graphics")
105   styles = graphs~XNameAccess
106   titelStyle = styles~getByName("headline")
107   styleProps = titelStyle~XPropertySet
108   RETURN styleProps~getPropertyValue("DisplayName")
109
110 ::requires UNO.CLS   -- load UNO support for OpenOffice.org
111
112 /* procedure for formating the agenda slide */
113 /* assigning of layout and heading */
114 ::routine  prepareAgendaSlide
115   use arg agendaPage
116
117   xPageName = agendaPage~XNamed
118   xPageName~setName("agenda")
119   agendaPageProps = agendaPage~XPropertySet
120   agendaPageProps~setPropertyValue("Layout", box("short", 1))
121
122   xShapes = agendaPage~XShapes
123   agendaHeadingShape = xShapes~getByIndex(0)
124   xText = agendaHeadingShape~XText
125   xText~setString("Agenda")
126
127 /* procedure for formating the listing */
128 /* increasing of line spacing and fontsize */
129 ::routine formatAgenda
130   use arg agendaShape
131
132   headingListShapeProsp = agendaShape~XPropertySet
133   lineSpacing = headingListShapeProsp~getPropertyValue("ParaLineSpacing")
```

```
134   lineSpacing~Height=500
135   lineSpacing~Mode=2
136   headingListShapeProsp~setPropertyValue("ParaLineSpacing", lineSpacing)
137   headingListShapeProsp~setPropertyValue("CharHeight", box("float", 40))
```

Snippet 20: Create an agenda (09_agenda.rex)

The last macro of this bachelor paper is a script that generates a slide with the agenda of the presentation. The agenda will be created using the text fields with heading style. If the agenda is already created in the past and the script is started again, the agenda will be updated with the current heading tags. This script will again use various parts of the previous examples to retrieve the array with the heading information. To ensure a failsafe process, the macro checks if there is any heading text field at all and shows a message dialog in the case of nonexistence.

```
112 /* procedure for formating the agenda slide */
113 /* assigning of layout and heading */
114 ::routine  prepareAgendaSlide
115   use arg agendaPage
116
117   xPageName = agendaPage~XNamed
118   xPageName~setName("agenda")
119   agendaPageProps = agendaPage~XPropertySet
120   agendaPageProps~setPropertyValue("Layout", box("short", 1))
121
122   xShapes = agendaPage~XShapes
123   agendaHeadingShape = xShapes~getByIndex(0)
124   xText = agendaHeadingShape~XText
125   xText~setString("Agenda")
```

This procedure gets a drawpage as an argument and sets the name of it for enabling the possibility to refresh it when running the macro at a later date. The *Layout* property is set to 1 that means the slide gets a text field at the top and a listing at the center. The text of the text shape is set to “Agenda” by accessing the shape with the index 0.

```
127 /* procedure for formating the listing */
128 /* increasing of line spacing and fontsize */
129 ::routine formatAgenda
130   use arg agendaShape
131
132   headingListShapeProsp = agendaShape~XPropertySet
133   lineSpacing = headingListShapeProsp~getPropertyValue("ParaLineSpacing")
134   lineSpacing~Height=500
135   lineSpacing~Mode=2
136   headingListShapeProsp~setPropertyValue("ParaLineSpacing", lineSpacing)
137   headingListShapeProsp~setPropertyValue("CharHeight", box("float", 40))
```

For formating the listing with the headings this procedure is called. The line spacing as well as the font size is adjusted to create a good looking agenda.

```
69 /* positioning of the text cursor */
70 headingListShape = agendaPage~XShapes
71 headingListShape = headingListShape~getByIndex(1)
72 xText = headingListShape~XText
73 xText~setString("")
74 xTextCursor = xText~createTextCursor
75 xTextCursor~gotoEnd(.false)
76 xTextRange = xTextCursor~XTextRange
```

The script appends each heading to the listing text field on the agenda slide by using a *TextCursor*. Line number 73 shows the command how the cursor will be placed at the end. The *XTextRange* interface will then be used to append the text with a carriage return.

Figure 28 shows a generated agenda slide.

Figure 28: Generated agenda

5 Conclusion

The conclusion of this bachelor thesis is that with the help of Open Object Rexx, Bean Scripting Framework, BSF4Rexx and OpenOffice.org it is possible to automate the office suite without any expert knowledge about programming languages. The only real requirement for the development is to be able to search self-dependent in the web for the functionality one is looking for. But this is not just the case for this type of work. For nearly every task one is coping with, there is a need to find the desired information as fast as possible, but not loosing the sight of the quality of the results.

The open source scripting language Open Office Rexx is very easy to learn and can use the massive amount of classes provided by Java with the help of BSF and BSF4Rexx. This will give the scripting language more power and opens the door to the UNO based architecture of OpenOffice.org.

The macros themselves are very useful and could save presenters a lot of time, especially for the macros, which are dealing with the progress indication. But that should not be the only important output of this paper. Other students who are also interested in this topic can benefit from the results and build open the snippets to create even more sophisticated macros.

Because there were not so many examples for automation of Impress, actually there is only one on the homepage of OpenOffice.org, it was not very easy to get in touch with the interfaces, properties and functionality of Impress. With the help of the excellent Developers Guide, some nutshells from previous bachelor papers and the online community of OpenOffice.org it was possible to create the snippets in this bachelor paper in a reasonable amount of time.
It also emerged that the functions in both the *UNO.CLS* and *BSF.CLS* were very helpful, for example the ones which return the names of the inferfaces using reflection and the properties of an object.

And if more and more snippets and examples are getting online, the community grows and grows and the collective knowledge would increase over the years, the power of OpenOffice.org and open source software in general will be strengthen.

6 References

[Aham05] Ahammer, Andreas: OpenOffice.org Automation: Object Model, Scripting Languages, „Nutshell“-Examples, Bachelor Course Paper, 2005

[BSF07a] Apache Jakarta Project: Bean Scripting Framework, Front Page, http://jakarta.apache.org/bsf/, retrieved on 2007-05-26

[BSF07b] Apache Jakarta Project: Bean Scripting Framework, Frequently Asked Questions, http://jakarta.apache.org/bsf/faq.html, retrieved on 2007-05-26

[BSF07c] Apache Jakarta Project: Bean Scripting Framework, Manual, http://jakarta.apache.org/bsf/manual.html, retrieved on 2007-05-26

[BSF4Re07a] Flatscher, Rony G.: The Vienna Version of BSF4Rexx - Changes for OOo, http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/changesOOo.txt, 2007-01-28, retrieved on 2007-05-24

[BSF4Re07b] Flatscher, Rony G.: The Vienna Version of BSF4Rexx – Reference Card OpenOffice.org, http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/refcardOOo.pdf, 2007-01-28, retrieved on 2007-05-24

[BSF4Re07c] Flatscher, Rony G.: The Vienna Version of BSF4Rexx – Readme of BSF4Rexx, http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/readmeBSF4Rexx.txt, 2007-01-28, retrieved on 2007-05-24

[BSF4Re07d] Flatscher, Rony G.: The Vienna Version of BSF4Rexx – Readme of OOo support, http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/readmeOOo.txt 2007-01-28, retrieved on 2007-05-24

[Flat05] Flatscher, Rony G.: Automating OpenOffice.org with OoRexx: Architecture, Gluing to Rexx using BSF4Rexx, 2005, Wirtschaftsuniversität Wien (Vienna University of Economics and Business Administration), Austria, http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf, retrieved on 2007-05-20

[Flat06a] Flatscher, Rony G.: Resurrecting REXX, Introducing Object Rexx, 2006, Wirtschaftsuniversität Wien (Vienna University of Economics and Business Administration), Austria, http://prog.vub.ac.be/~wdmeuter/RDL06/Flatscher.pdf, retrieved on 2007-05-18

[Flat06b] Flatscher, Rony G.: Automatisierung von Windows Anwendungen (1) - Einführung, Überblick, Anweisungen, Prozeduren, Funktionen, http://wi.wu-wien.ac.at/rgf/wu/lehre/autowin/material/folien/Automatisierung_01.pdf, retrieved on 2006-03-15

[Flat06c] Flatscher, Rony G.: The Vienna Version of BSF4Rexx, Presentation at the 2006 International Rexx Symposium, USA, 2006, http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf, retrieved on 2007-05-20

[Flat06d] Flatscher, Rony G.: Automatisierung von WindowsAnwendungen (3) – Ausnahmen (Exceptions), Referenzen, Direktiven (::routine, ::requires) http://wi.wu-wien.ac.at/rgf/wu/lehre/autowin/material/folien/Automatisierung_03.pdf, retrieved on 2006-03-17

[OORexx07a] Open Object Rexx Homepage, Product Brochure, http://www.oorexx.org/ooRexx-Brochure.pdf, retrieved on 2007-05-17

[OORexx07b] Open Object Rexx Homepage, About, http://www.oorexx.org/index.html, retrieved on 2007-05-17

[OpenOf07a] Open Office Homepage, About , http://about.openoffice.org/index.html, retrieved on 2007-05-20

[OpenOf07b] Open Office Homepage, Product , http://www.openoffice.org/product/index.html, retrieved on 2007-05-20

[OpenOf07c] Open Office Homepage, API – Developers Guide, First Steps, http://api.openoffice.org/docs/DevelopersGuide/FirstSteps/FirstSteps.xhtml, retrieved on 2007-05-20

[OOoDev05] Open Office Homepage, Developers Guide, May 2005, http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf, retrieved on 2007-05-214

[WikiBS07] Wikimedia Foundation Inc, Bean Scripting Framework, http://en.wikipedia.org/wiki/Bean_Scripting_Framework, retrieved on 2007-05-06

[WikiOo07] Wikimedia Foundation Inc, OpenOffice.org, http://en.wikipedia.org/wiki/Openoffice.org, retrieved on 2007-05-05

[WikiRe07] Wikimedia Foundation Inc, REXX, http://en.wikipedia.org/wiki/REXX, retrieved on 2007-05-04