ooRexx Snippets for OpenOffice.org Writer

Matthias Prem
Vienna University of Economics and Business Administration
Reg. No. 0252896
E-Mail: h0252896@wu-wien.ac.at

Version 2.1: 2006-07-24

Bachelor Course Paper
Department of Business Informatics
Prof. Dr. Rony G. Flatscher
Department Chair "Information Systems and Operations"

Table of Contents

List of Figures

1 Introduction

1.1 About this Paper

This paper deals with the use of ooRexx as a scripting language for automation of OpenOffice.org Writer.

At first, there will be an introduction to the technical requirements, which include the software that has to be installed. Concerning ooRexx there is also a sub chapter about its syntax and common instructions, to give a feeling for this programming language.

The next chapter is about the architectural approach behind ooRexx and OpenOffice.org. It is described how OpenOffice.org can be accessed using ooRexx.

Chapter four is a small installation guide, which shows how to set up the different software programmes and configure them correctly.

Chapter five and six show how the automation of OpenOffice.org Writer can be done. Small snippets, which are code examples, demonstrate different tasks.

At last the conclusion gives a small summary and an outlook.

1.2 Research Question

How can the OpenOffice.org Writer be automated using ooRexx?

2 Technical Requirements

2.1 OpenOffice.org

2.1.1 History

OpenOffice.org was created originally by a german company called StarDivision. Back then its name was StarOffice. When StarDivision was acquired by Sun Microsystems OpenOffice.org was born. [Open04]

In 2000 Sun published the Source Code under several licenses and wanted to build a community around the software, which should improve it on an open-source basis. This new project was known as OpenOffice.org. [Wiki01]. Looking at the homepage of OpenOffice.org and all its user contributed sites, documentations and tools it quickly comes clear that Sun's plan to build a community turned out to be a big success.

Sun now builds StarOffice on the source code of OpenOffice.org and adds extra features to it. Since October 2005 OpenOffice.org is available in version 2.0, where the latest stable release is 2.0.2. [Wiki01]

2.1.2 Overview

OpenOffice.org is an office suite like Microsoft Office on Windows Computers, or iWork on Macintosh or Koffice on Linux Computers. OpenOffice.org is available in many different languages, even in Esperanto [Open01][1], and supports different platforms. The main differences between e.g. Microsoft Office and OpenOffice.org are the unrestricted availability of OpenOffice.org and its state of an open-source project.

OpenOffice.org consists of mainly these parts:

- OpenOffice.org Writer which is used for creating text documents. [Open03]

[1] For a full list of supported languages see: http://projects.openoffice.org/native-lang.html

- OpenOffice.org Calc used for creating tables and doing calculations concerning tables, cells and formulas. [Open03]
- OpenOffice.org Base is used for database applications. [Open03]
- OpenOffice.org Impress is the tool for making powerful and, as the name suggests, impressive presentations. [Open03]
- OpenOffice.org Draw is used to create drawings. Everything from simple diagrams to powerful 3D illustrations is possible. [Open03]

What is especially mentionable about OpenOffice.org, is that it doesn't use a proprietary file format but since version 2.0 the OASIS OpenDocument format. [Open3]

The goal of the OASIS OpenDocument Standard is to provide a file format, which can be read by any application, which understands XML[2]. [Oasi01]

Therefore, it is a possible way out of a lock-in effect resulting by the use of vendor-dependent file formats like the `*.doc` used by Microsoft Word.

2.2 Open Object Rexx

2.2.1 History

The original Rexx was a scripting language implemented by Mike Cowlishaw of IBM in 1982. The purpose of Rexx was to have a scripting language for any system, which was supported by IBM. IBM made it available on nearly every of its operating systems, like VM/CMS or AS/400. [Wiki02]

In the 1990 there where two new Rexx versions:

- NetRexx, which compiles to Java bytecode[3] and is not compatible to the original Rexx. [Wiki02]
- Object Rexx is an object-oriented version, which is compatible to the original Rexx. [OoRx04]

[2] Extended Markup Language, a definition language used to describe the structure of a document.
[3] see chapter 2.3.2 for an explanation

After IBM released Object Rexx under the Common Public Licence in October 2004, the first Open Object Rexx was given to the people a few months later in February 2005. [Wiki02]

In 1990 the first independent Rexx symposium took place. A result of this symposium was the founding of the Rexx Language Association. [Wiki02] The goal of the RexxLA is to promote the use of Rexx and all its variants as well. [RxLa01]

2.2.2 Overview

Open Object Rexx is an open source programming language, which is easy to use and learn but nevertheless powerful due to the following features:

- The use of a human oriented language, in ooRexx's case English. Typical keywords are SAY, PULL, DO. [OoRx01]
- ooRexx has few rules. Normally there is one command per line, but separated by an semicolon is is possible to have more commands in one line. Also one command can span more lines. A semicolon at the end is not obligatory, but it doesn't hurt either. It is also allowed to use names of built-in functions as own variables, ooRexx doesn't get tangled, it recognizes which one is meant. [OoRx01]
- ooRexx is interpreted, which means that there is no need for a compiler. The Rexx code is read line by line and run by the processor. This feature speeds the development time up. [OoRx01]
- A rich set of functions and methods is already implemented in OORexx including searching, comparing text and numbers and also performing arithmetic calculations. [OoRx01] As we will see later on with BSF4Rexx this library of functions and methods even gets bigger.
- To a programmer used to variable declarations the concept of typeless variables in ooRexx might seem a little strange. ooRexx recognizes the type of the variable and, provided it is possible, performs the requested function on it. [OoRx01]

- The string handling offered by ooRexx is very powerful. It is easy to manipulate strings or characters. [OoRx01] The PARSE function allows very quickly to read important parts of strings and drop unnecessary parts.[4]
- Unlike other programming languages, which use binary arithmetic, ooRexx uses decimal arithmetic. The advantage of this concept is that decimal arithmetic is more accurate, as it is the human way of calculating. [OoRx01]
- When a script throws an error the error message offered by ooRexx is explaining the problem in high detail, which makes the solving much easier. Using the TRACE instruction the programmer has got a sophisticated debugging tool at his hand. [OoRx01]
- ooRexx is, as the name suggests, object oriented. This means that it supports classes, objects and methods. [OoRx01]
- ooRexx can be used as a scripting language for the Windows operating system. [OoRx04]

2.2.3 Syntax[5]

The following paragraphs will provide a short introduction into the Syntax of ooRexx, which is necessary to understand the Snippet examples in detail.

Starting with a very simple example, we will learn about the very basic functions SAY and PULL.

```
/* PULL something and SAY it */
SAY 'Please enter your name:'
PULL name
SAY 'Hello there,' name'!';
EXIT 0
-- The End
```

Running the Script gives the following output:

```
C:\rexx saypull.rex
Please enter your name:
yogi-bear
Hello there, YOGI-BEAR!
```

[4] see [OoRx5] for information on the PARSE function of ooRexx
[5] for more information on ooRexx see [OoRx05] and [OoRx04]

So, what can we see in the example above?

- A comment can either be between a slash asterisk combination, where it can span more than one line, or be after two minus, where it is in one line.
- A line can be ended with an semicolon, but it is optionally.
- The string “yogi-bear” is “pulled” to the variable (which needs not to be declared) and printed on the screen[6].
- For printing a given text on the screen it has to be encased by inverted commas, while the variable “name” must not be encased to be interpreted correctly.
- The `EXIT` instruction ends the Rexx program, it is, as the semicolons, optional. It can return a value (in our example 0 is returned).

The next example will show how Rexx handles variables.

```
/* Rexx and variables */
a = 5; b = 6; c = a + b
d = 'String01'; e = 'String02'; f = d || e
SAY a b c; SAY d e f
EXIT 0
```

```
D:\>rexx test.rex
5 6 11
String01 String02 String01String02
```

- As you see, there is no need to declare the variables.
- Rexx performs the operations fine. `a + b` results in a number and `d || e` in a concatenation.
- Trying to do `d + e` would, of course, result in an error, as an arithmetic operation on strings is not allowed.

Now let's look at how loops and routines are realized in Rexx, using the following two scripts `calling.rex` and `routine.rex`.

[6] `PULL` reads the text and converts it to upper-case. If you want to avoid this, you have to use the `PARSE PULL` function instead [ooRexx05].

```
/* calling.rex */
result=multiply(12,2)
SAY result

::REQUIRES routine.rex
```

```
/* routine.rex */
::ROUTINE multiply PUBLIC
result = 0

USE ARG x, b
DO b
  result = x + result;
END
RETURN result
```

- What happens is, that `calling.rex` calls a routine which is found in `routine.rex`.
- `routine.rex` is included in `calling.rex` using the requires command. `calling.rex` looks for the routine multiply in `calling.rex` and `routine.rex`.
- In `routine.rex` a simple task is performed using a loop, which is done two times in that case. The result is returned to `calling.rex`.

In the next code example, where we are going to travel to Sweden, we will do the step from Classic Rexx to Object Rexx.

Here's how to handle `CLASS` and `METHOD`:

```
/* travel.rex */
country1 = .country~New;
country1~name = "Austria"
country1~temperature = 18
country1~weather = "rainy"

country2= .country~New("Egypt", 35, "sunny")
country2= .country~New("Sweden", 15, "cloudy")

country1~countryinfo
country2~travelthere;
country2~travelthere;

EXIT 0

::CLASS country
::METHOD INIT
    EXPOSE name temperature weather
    USE ARG name, temperature, weather
    self~tourist=0
::METHOD name ATTRIBUTE
::METHOD tourist ATTRIBUTE
::METHOD temperature ATTRIBUTE
```

```
::METHOD weather ATTRIBUTE
::METHOD countryinfo
    SAY "About the country:"
    SAY "Name:" self~name; SAY "Temperature:" self~temperature
    SAY "Weather:"  self~weather
::METHOD travelthere
    SAY "You travel to " self~Name"."
    self~tourist=self~tourist+1
    SAY "Number of tourists: "self~tourist
::METHOD leave
    SAY "You've left "self~name"."
::METHOD UNINIT
    SAY "You can't travel to" self~name "any longer."
```

Looking at the code above we can learn the following:

- A class is defined using the `::CLASS` tag followed by a name of the class.
- When creating a new instance of a class using `country1=.country~New`, `::METHOD INIT` is called, where there are two ways to tell ooRexx the values of the object's attributes. One way is to declare the values one by one as it is done for the country Austria. The other way is to send arguments to `::METHOD INIT` as it is done for Sweden and Egypt.
- Attributes of a class are declared using `::METHOD <name of attribute> ATTRIBUTE`.

The different methods itself perform the following tasks:

- `::METHOD countryinfo` requests the attribute-values of the country by `self~<name of attribute>` and prints them on the screen.
- `::METHOD travelthere` sends a tourist to the designated country and increments the total number of tourists using again the `self~` tag.
- `::METHOD leave` should be self explaining.
- `::METHOD UNINIT` needs not to be called explicitly. It is called whenever the script is finished.

The crucial thing to learn from our travelling example is how to send messages to objects. `::METHOD travelthere` for Sweden is invoked by `country2~travelthere`. The most important sign in this case is the so

called “twiddle”, which is the “~” sign. In ooRexx the twiddle is the message operator[7], used to send messages to objects. We will need it in virtually every snippet example.

2.3 Java

2.3.1 History

In comparison to programming languages like C or Pascal, Java is quite a young programming language. [Niem02]

At the beginning of the 90ies a project team at Sun Microsystems called „Green Project“ aimed at developing a programming language for the electronic consumer area. The first prototype was called Oak[8]. Small programmes, written in this language, were intended to be running in household utilities like refrigerators, tv-sets or telephones. [Niem02]

When experiencing problems with the market, as there was less interest in this new technology as predicted the new language was nearly bound to die. [Niem02]

In 1993, when the Internet became more important and grew bigger, Sun recognized, that it is a good field for the Oak programming language. It was a platform independent language and so ideal for the use in the heterogeneous Internet. [Niem02]

Where the name „Java“ derives from is subject of many assumptions. One possible theory for the choice of the name is that „Java“ is used in American English as another word for coffee. As coffee was the favourite drink for the Sun programmers the choice of the name was obvious. [Niem02]

[7] In Java the message operator is a „.“.
[8] Object application kernel [Niem02]

2.3.2 Overview

Java is, as ooRexx can be, an object-oriented programming language. In ooRexx the programmer can decide, whether he wants to write his scripts object-oriented or procedural (Classic Rexx).

In Java the programmer has to stick to the the object-oriented paradigm, there is no choice for him. [Stey01]

Another difference between ooRexx and Java is the way of executing the programmes. As we learned before, ooRexx is interpreted by the interpreter and there is no need for compiling. Java source code has to be compiled to byte code and then run by the interpreter. [Niem02]

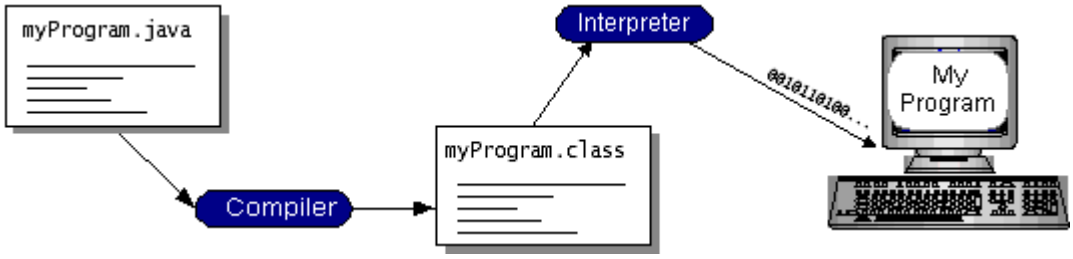

Figure 1: How Java is compiled and interpreted [Java01]

Once Java source code is compiled into byte code it can be run by the interpreter on any operating system, where Java is available. The byte code is understandable code for the Java Virtual Machine. [Niem02]

The Java Virtual Machine can be seen as mediator between the byte code from the compiler and the machine code the processor understands. The Java Virtual Machine is available for nearly every operating system and therefore makes Java platform-independent. [Niem02]

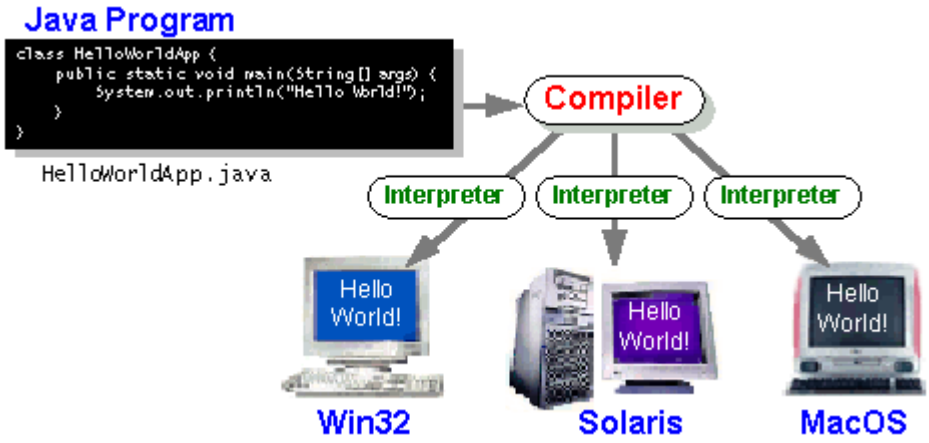

Figure 2: Java byte code can be executed on any platform [Java01]

The most important fact about Java for our work is, that it is developed by Sun Microsystems. As we learned before StarDivision was bought by Sun and now Sun is developing StarOffice and accordingly OpenOffice.org. Having this two products in responsibility of Sun makes this work possible at all, as we will see in the next chapters.

Java comes in different versions, which include:

- Java SE: Java Standard Edition which is the product of choice for a standard user. You can decide if you want the Java Development Kit (JDK), if you want to write your own Java programmes or the Java Runtime Environment (JRE), that allows you to run Java programmes, but has got less support for developing. [Java02]
- Java EE: Java Enterprise Edition includes a Java Application Server, helps developing Java-based web services and much more. [Java03] The target group for this Java product are software developers in companies, who want to create online-shops, ERP[9] software or CRM[10] software.
- Java ME: Java Mobile Edition is designed for different parts of small devices. [Java04] This includes mobile phones, handheld devices and others. Java ME is therefore a tool of choice for M-Commerce[11].

[9] Enterprise Ressource Planning
[10] Customer Relationship Management
[11] M-Commerce: Mobile Commerce, commerce made using mobile devices

2.4 BSF

Before talking about BSF4Rexx we have to learn something about the Bean Scripting Framework in general, to gain a deeper understanding about how it works in general which will help us explaining BSF4Rexx.

2.4.1 Overview

A Bean Scripting Framework is a set of Java classes, which allows the use of scripting languages in Java applications and access to Java classes and functions from scripting languages vice versa. For example BSF makes it possible to write JSPs[12] using a scripting language [JBSF01] but not loosing the potential of Java's functions and methods.

The year of birth of BSF is 1999, where the initial intent was to access JavaBeans[13] from scripting languages. Moving from IBM's T.J. Watson Research Center[14] to IBM's AlphaWorks[15] developer site BSF became an open source project. As BSF became part of the Apache project Xalan[16] it was subordinated to the Apache Jakarta[17] project in 2002 as a gift from IBM to the Apache Software Foundation. The current version of BSF is 2.3. [JBSF01]

At the moment BSF supports e.g. Javascript, Python and Tcl. Some languages need their own BSF engines (e.g. JRuby, JudoScript) [JBSF02].

So, why is it called “Bean” Scripting Framework?

The FAQ gives us the following answer:

> „It's the beanage. Beans were the cool thing when BSF was first being designed, and BSF contains several flavors.“ [JBSF01]

---

12 Java Server Pages, used to generate dynamic Webcontent

13 JavaBeans are Java programs, which can be assembled to form a bigger application. [JBSu01]

14 IBM's Thomas J. Watson Research Center is the headquarter of the IBM Research Division, which is the largest industrial research organization. [IBMW01]

15 IBM's AlphaWorks provides developers with the newest innovations from IBM, so that they themselves can help to improve them. [IBMA01]

16 The goal of the Apache Xalan project is to provide XSLT support on different platforms, which combines XML with XPath (allows to address XML objects directly). The Apache Xalan Project is no longer part of the Jakarta Project, but has become an independent one. [ApXa01]

17 The Jakarta Project offers different open source Java solutions and is part of the Apache Software Foundation, which intents to support the collaborative development of open source software. [ApJa01]

Asking www.urbandictionary.com gives us the answer, that “beanage” refers to something pretty cool, and that's what BSF actually is.

2.4.2 Architecture

The Bean Scripting Framework consists of mainly two parts:

- BSFManager
- BSFEngine

It is the job of the BSFManager to allow access to Java objects and manage the scripting execution engines. A Java application can get access to the BSFManager class by creating an instance of it. [JBSF03]

The BSFEngine needs to be implemented for each language, which is indented to be used by BSF. The engine abstracts from the scripting language and handles the script execution and object registration. [JBSF03]

It is possible for an application to use one BSFManager through different BSFEngines. So different script languages use the same BSFManager. Once objects are registered with the BSFManager they stay alive until the BSFManager is shut down again. [JBSF03]

2.4.3 BSF4Rexx

As the name suggests BSF4Rexx is a Bean Scripting Framework for the Rexx language.

The first proof of concept was done in 2000/2001 by a student named Peter Kalender. As he was a student of the University Essen the first BSF4Rexx version was named „Essener Version“. Work went on and in Spring 2001 BSF4Rexx was presented to the RexxLA. Two years later a new version of BSF4Rexx, called the „Augsburger Version“, was introduced. It had some bugs fixed and external Rexx functions added, including the loading of Java on Windows and Linux platforms. [Flat01]

The recent version of BSF4Rexx is the „Vienna Version“. It includes not only the necessary files for working with BSF4Rexx but also a setup routine, making it easier to install the package and use it.

The next two code examples show how to include BSF4Rexx into a Rexx script.

```
/*Using call for including BSF support*/

CALL BSF.CLS

.bsf.dialog~messagebox("This is a Java message box.")
SAY "Did you see it?"

EXIT 0
```

```
/* Using require for including BSF support */

.bsf.dialog~messagebox("This is a Java message box.")
SAY "Did you see it?"

EXIT 0

::REQUIRES BSF.CLS
```

As you see, there are two ways to include `BSF.CLS`, which enables the BSF support for Rexx:

- BSF support for Rexx is included using either `CALL BSF.CLS`. It has to be before the first use of BSF, otherwise it will result in an error.
- Using `::REQUIRES BSF.CLS` is another way, we already know from chapter 2.2.3 were we learned, how to include a Rexx Scripts, that provided us with a multiply function. `::REQUIRES` has to be at the end of the script. It is invoked at the beginning of running and therefore the functions are available throughout the whole script. That is the reason, why we are going to use the `::REQUIRES` command in the future.

The comparison of `routine.rex` and `BSF.CLS` is quite a good illustration. Where `routine.rex` gives us one routine named `multiply` to access, `BSF.CLS` gives us a whole bunch of them. And so does Java give Object Rexx even more functions.

The actual event is invoked by `.bsf.dialog~messagebox("This is a Java message box.")`. It shows a message box, with the entered text accompanied by a beep. When looking into the documentation we will see, that

the class `BSF.DIALOG` provides the methods `messageBox()`, `dialogBox()` and `inputBox()` using the class object `.BSF.DIALOG`. A message box can be called using `messageBox(message, [title], [type])`, where `[title]` and `[type]` are optional. [Flat01]

The last statement is a `SAY`, to illustrate, that we are still speaking the Rexx language, so there is no need for a `System.Out.Println`[18].

The Beep signal is something one might find strange, as there is normally no beep, when calling a message box from Java. Looking into `BSF.CLS` helps us finding an answer.

`::method messageBox` is a method of the `bsf.dialog` class as described above. In the `BSF.CLS` it looks like this:

```
::class bsf.dialog public

<<snip>>

::method messageBox
  expose component
  use arg message, title, messageType
  dlgConstants = self~class~dlgConstants -- get dlgConstants
  javaDialogClass = self~class~javaDialogClass -- get Java class object
  call beep 1500, 100 -- beep in an attention tone
  if arg()=1 then -- only one argument given
  do
    javaDialogClass~new~showMessageDialog(component, message)
  end
```

As one can see is, that the `::method messageBox` uses different arguments to appear on the screen. We already know that from our travel example. If there is only one argument, which is the case, `javaDialogClass~new~showMessagDialog(component, message)` is called. `component` is an attribute described in `::method init` and represents the responsible Java component for execution.

The important thing we are looking for is in the middle of the code. There it says: "`call beep 1500, 100`". That's the command, responsible for the unusual beep we hear, when executing our script.

[18] System.Out.Println is the Java command to print something to the standard output, which is normally the screen.

3 Bringing together OpenOffice.org and BSF4Rexx

3.1 The Architecture of OpenOffice.org

3.1.1 UNO – The base Technology of OpenOffice.org

UNO means Universal Network Objects and is, as the title is suggesting, the base technology of OpenOffice.org. It enables the user to write components which work across different programming languages and even across different operating systems. [Open05]

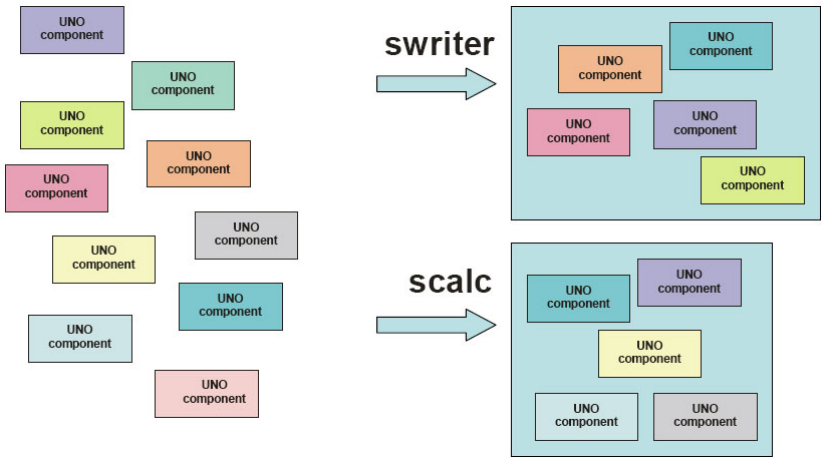

Figure 3: The different parts of OpenOffice.org consist of several UNO components [Flat02]

UNO can be used by programming languages like C++ or Java and also by scripting languages like JavaScript or Jython. Using the Common Language Infrastructure Binding it can also be accessed with .NET languages. [Open05]

The API[19] gives UNO access to OpenOffice.org and describes its programming features as well. [Open05] The API can be viewed at: http://api.openoffice.org/ and can be often a major help when looking for solutions.

UNO also allows to be used for crossing different networks. It is possible to connect to a local or a remote instance of OpenOffice.org. [Open05]

[19] Application Programming Interface

To make this feature available OpenOffice.org has to be started in listening mode by modifying the `Setup.xcu` file like this[20]:

```
<prop oor:name="ooSetupConnectionURL">
```

has to be replaced by:

```
<prop oor:name="ooSetupConnectionURL">
<value>
  socket,host=localhost,port=2002;urp;StarOffice.ServiceManager
</value>
</prop>
```

UNO use bridges to send and receive between different components and objects using a protocol called URP[21], which is supported by either TCP/IP[22] sockets[23] or pipes[24]. Both ends of the URP has to be an UNO environment, meaning that any of the UNO supported languages is necessary. [Open05]

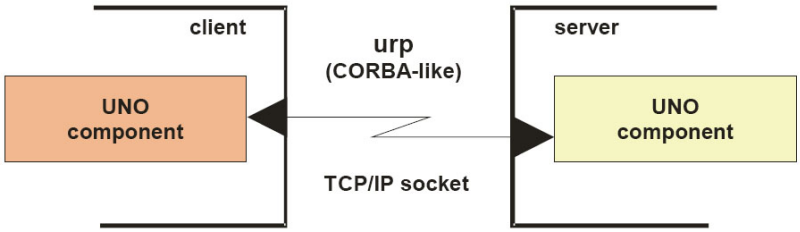

Figure 4: Communication between two UNO components [Flat02]

The last parameter of the code above leads us right to the next topic:

3.1.2 The Service Manager and Services

The Service Manager is a typical concept of UNO. It can be seen as a factory, which provides services. In a very simple way, services can be seen as UNO objects, performing a specific task. [Open05]

Two typical services are by example:

- `com.sun.star.Desktop` which is used to load documents, access loaded documents, and so on. [Open05]

[20] for more information: see [Open05] Chapter 3.3.1 UNO Interprocess Connections
[21] UNO remote protocol, communication protocol that is used by UNO components
[22] Transmission Control Protocol/ Internet Protocol, the two most used protocols on the Internet today.
[23] A socket is defined by an ip adress, a port number and a protocol and is used in network communications. [Open05]
[24] A pipe lets two processes communicate with each other. Pipes are faster than sockets, but only work, if the two processes are running on the same machine. [Open05]

- `com.sun.star.GlobalSettings` is used to manage global view or printer settings. [Open05]

A service always has to be in a component context. The component context consists of the service manager and other data used by the service. [Open05] The component context we are going to use is the Writer application.

3.1.3 Objects

In the context of OpenOffice.org an object is a software artefact, that has got methods we can call and attributes we can read and set. The methods and settings of an object are defined by its interfaces. [Open05]

3.1.4 Interfaces

As we heard before, an interface is a set of attributes and and methods belonging to an object. Actually interfaces would not be needed to set and read properties, as there could be used methods for this tasks. [Open05]

There are however two good reasons for using interfaces:

- It allows combination of getting and setting values, which is needed by many developers. [Open05]
- The designer of an interface can easier express the differences between objects. Attributes are used for non-integral parts of an object, while methods always go with core features. [Open05]

3.2 UNO.CLS

When talking about the `UNO.CLS` file, which comes with the BSF4Rexx package, we have to bear two things in mind:

- OpenOffice.org allows to be automated/controlled using different programming languages including Java.
- BSF4Rexx enables us to use Open Object Rexx to call Java functions.

Now we can conclude, that OpenOffice.org can be automated using ooRexx.

Since BSF4Rexx allowed access to Java it was possible to address OpenOffice.org using ooRexx.

In [Burg05] you will find an example, which illustrates this process, using the `::requires BSF.CLS` command `BSF.CLS` was included and having more than 20 lines of code, it was possible to connect to OpenOffice.org and open a new Writer document.

`UNO.CLS` helps the programmer, giving him easier access to OpenOffice.org supporting him with a number of standard routines, so he doesn't have to write them multiple times.

Although only the `UNO.CLS` is included in our scripts, it is easy to find out, that `BSF.CLS` is still in use, of course. Looking into the `UNO.CLS` file, we will find a line that says:

```
::REQUIRES BSF.CLS      -- get BSF4Rexx support
```

It makes clear that `UNO.CLS` uses the `BSF.CLS` library for accessing Java.

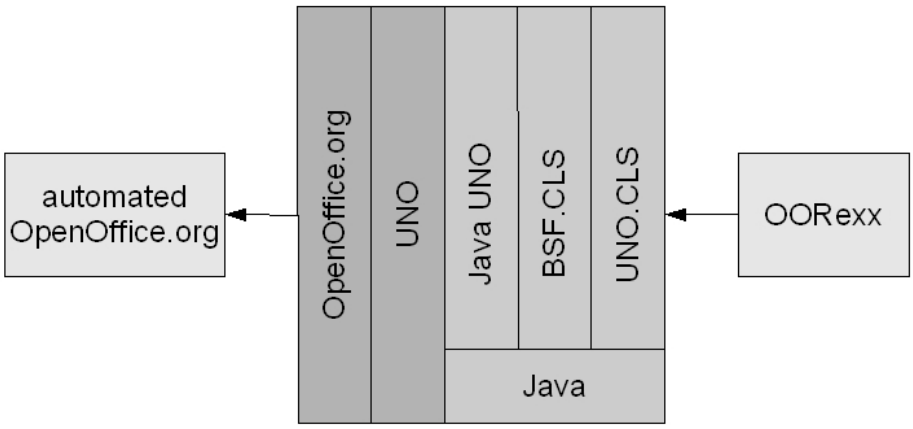

Figure 5: From ooRexx to OpenOffice.org (following [Augu05] cited in [Aham05])

This figure illustrates, how the commands are processed using BSF4Rexx. First from a ooRexx script `UNO.CLS` is called, which calls `BSF.CLS`. `BSF.CLS` uses its connection to Java to address the Java UNO model. Afterwards the OpenOffice.org UNO model sends the requests to OpenOffice.org and the result is an automation of OpenOffice.org. [Aham05]

4 Installation Guide

The following sub chapters will give an overview of the installation of the required software, which is needed to run the Snippets later on.

4.1 Download and install OpenOffice.org

The newest version of OpenOffice.org at the time of writing is 2.0.2 and is always available at: http://www.openoffice.org. Just download the free office suite in the language and operating system of your choice and install it.

It is also possible to order a CD, which includes a clipart collection, documentation and more. [Open07]

The documentation project is a good choice for learning more about the general features of OpenOffice.org. Opening the page http://documentation.openoffice.org/ in your web browser gives you a huge number of different papers including, how-tos, tutorials, user guides, setup guides and so on.

4.2 Open Object Rexx

Open Object Rexx comes in two versions at the moment. The stable version is 3.0.0, but there is also the 3.1.0 beta version available. [OoRx02] It's up to the user, which version he downloads and installs, the examples later on will run with both versions.

Open Object Rexx 3.0.0 and 3.1.0 beta for different operating systems can be downloaded at: http://www.oorexx.com.

4.3 Java

Before downloading and installing Java it is advisable, to check if there is Java already installed on your machine.

On both, Windows machines and Linux machines entering

```
c:\java -version
user@mymachine:/$ java -version
```

will give an output like

```
java version "1.5.0_04"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_04-b05)
Java HotSpot(TM) Client VM (build 1.5.0_04-b05, mixed mode, sharing)
```

informing the user, that there is a Java environment already installed.

If the Java version installed is at least 1.4, it is ready for being used with BSF4Rexx, so there is no need for a new install. [BSFR01]

Other outputs, including wrong version numbers or errors, require the user to install Java. He can decide between the JDK or the JRE, both versions are good and will allow running bsf4rexx. The recent version of Java is available at http://java.sun.com/j2se/1.5.0/download.jsp.

4.4 BSF4Rexx

BSF4Rexx is the last tool we need. Its current version is always available at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/. The install zip-file comes with the required files for installing and also with many examples for the use of BSF4Rexx and OpenOffice.org.

After unzipping the `BSF4Rexx_install.zip` file into your designated folder you have to run `setupBSF.rex` . This will create several files including `installBSF4Rexx.rex`. After executing it, you should be able to start BSF4Rexx scripts using `rexxj.cmd` on windows and `rexxj.sh` on Linux. [BSFR01]

Running either `infoBSF.rex` or `infoBSF-oo.rex` using `rexxj` or `rexx` should result in a reasonable output informing you, that BSF4Rexx is installed properly.

For detailed installation instructions or when having problems, view the `readmeBSF4Rexx.txt` file which comes with the BSF4Rexx installation package, or can always be found at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/readmeBSF4Rexx.txt

4.5 Setting up OpenOffice.org

After modifiying the `Setup.xcu` file as explained in chapter 3.1.1[25] we have to add OORexx Script support to OpenOffice.org.

Before enabling BSF4Rexx support in OpenOffice.org we have to enable a Java Runtime Environment for to be used by OpenOffice.org. Having a German version of OpenOffice.org you can do so by clicking: „Extras → Optionen“ or in an English version click „Tools → Options“.

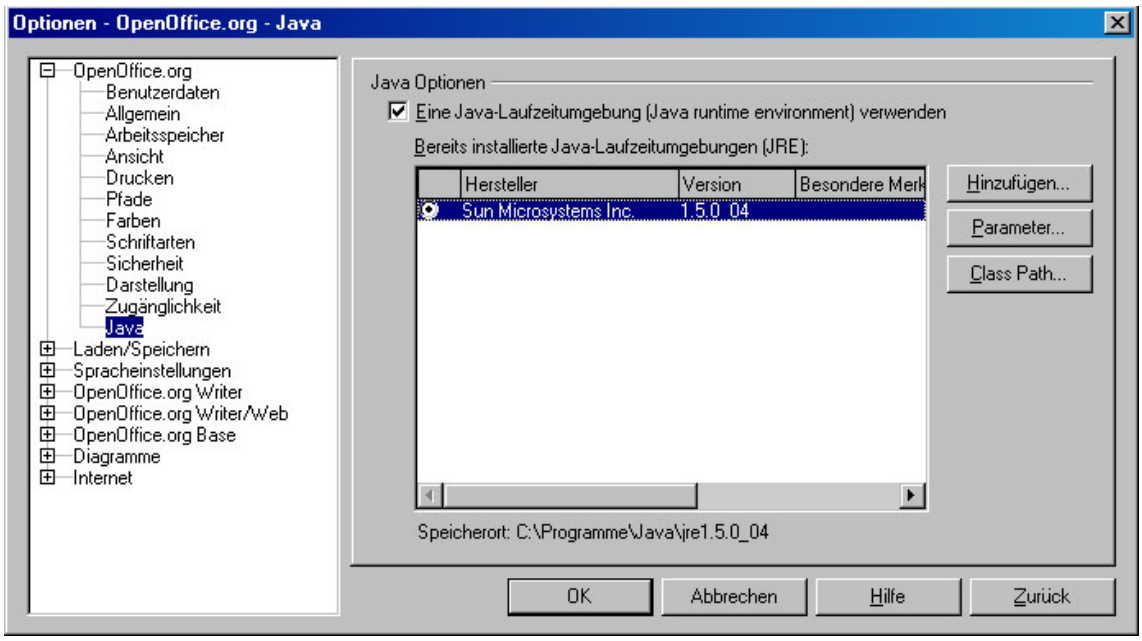

Figure 6: Enabling JRE in OpenOffice.org

In the opened window select „OpenOffice.org → Java” and click the checkbox to use a Java Environment.

In case you have got more than one JREs in the list above, make sure to select the same one, as BSF4Rexx was configured to. [Flat02]

It will usually be the one, that comes up, when entering `java -version` at the prompt.

Providing you have got OpenOffice.org 2.0 you can do the next step from inside OpenOffice.org, namely enabling the script support. Choosing „Extras →

[25] A more detailed explanation to changing the `Setup.xcu` file can be found in the Developer's Guide, which is available at: http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf

Package Manager“ or „Tools → Package Manager“ you get a window, where you can add additional packages to your OpenOffice.org installation. Choose `ScriptProviderForooRexx.jar` from your BSF4rexx directory, restart OpenOffice.org and you are ready to run the scripts.

Once you have performed these steps you will have the additional option „ooRexx“ in „Extras → Makros → Makros verwalten“ in German versions. In English versions the click path would be: „Tools → Macros → Organise → Macros → ooRexx“ [Burg05]

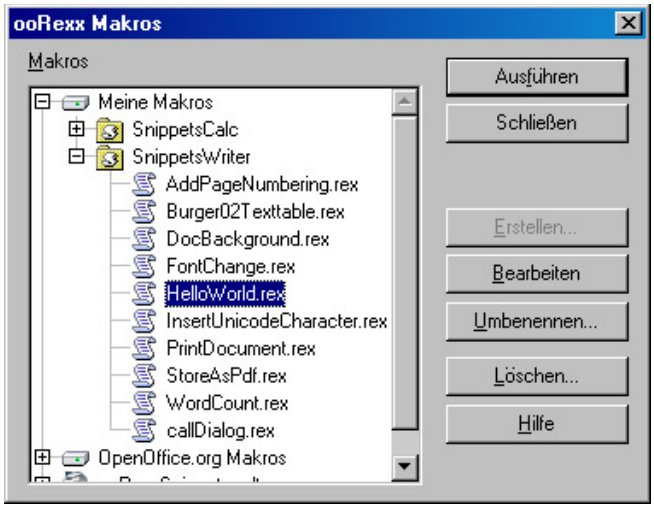

Figure 7: The ooRexx macros window

This is the point, where we will start writing some of our scripts later on.

5 Automating OpenOffice.org Writer

5.1 Working with Text Documents

A text document is a document model, which can handle text content. In our context a document can be stored and printed to make it permanent. The term „model“ refers to something that forms the basis of the document. It defines the document independently from its visual presentation. [Open05]

We have to be aware of the fact, that every time, when we want to talk to the model, we have to call it directly. In contrast, when we want to alter the visual presentation we have to use the controller of the document model. [Open05]

The two typical purposes of the controller are:

- Interacting with the user interface. This includes moving the visible text cursor, flipping pages on the screen or zooming in and out. [Open05][26]
- Providing information about the current selection, the current page or the total page count. [Open05]

The text document model consists of different architectural areas, which can be described as follows:

- The **text** is the core of the text document model. Its main parts are character strings, that are organized in paragraphs and other text contents. [Open05]
- The **service manager of the document model** must not be confused with the service manager, that is needed when connecting to OpenOffice.org for the purpose of automation. For each document model there is an own service manager. [Open05]

[26] see `AlterZoom.rex` for an example

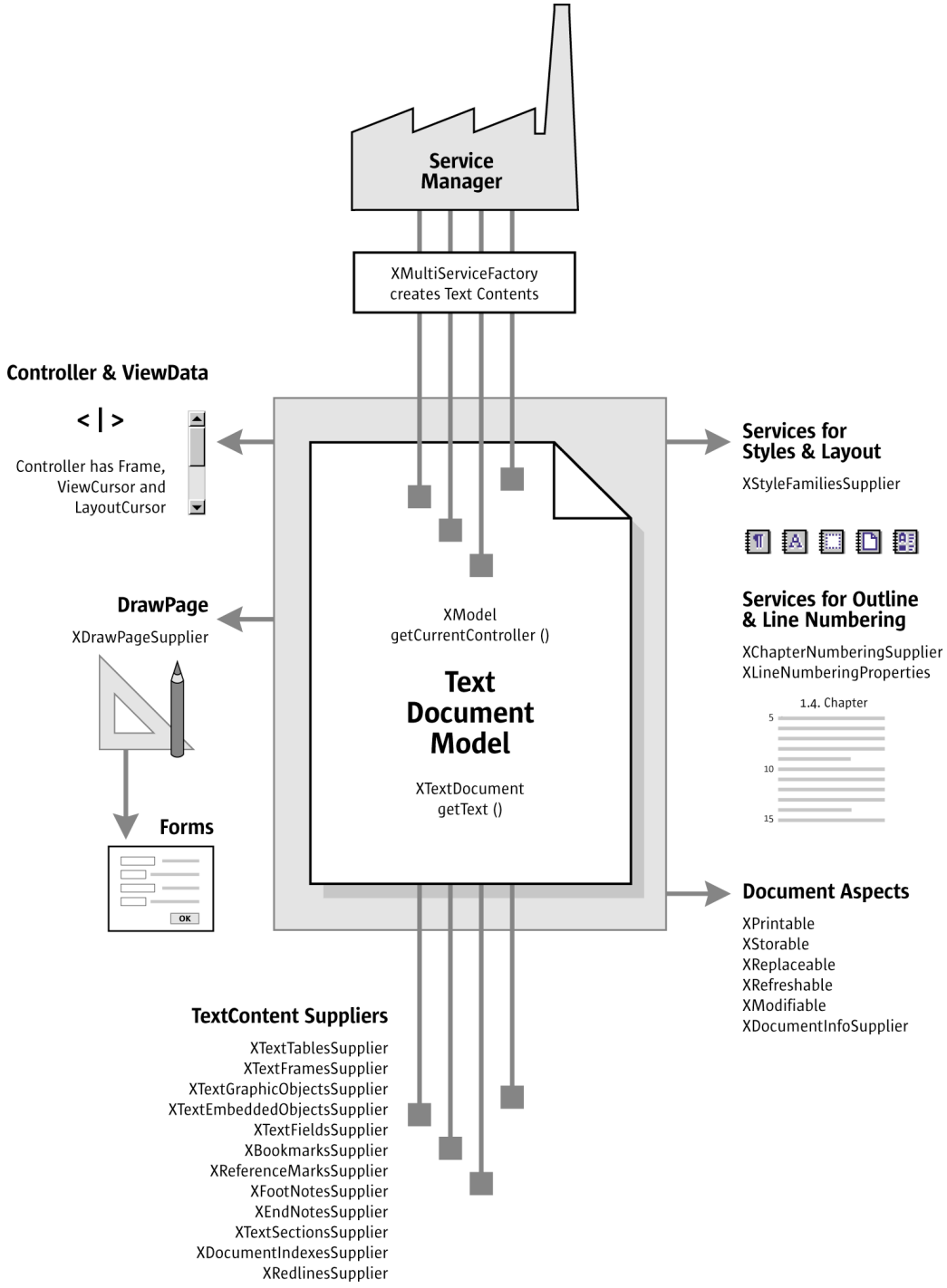

Figure 8: The text document model [Open05]

- Drawing shapes can not be found in the service manager of the document model, but can be found on the **draw page**, which can be considered as a transparent layer over the text. The draw page can affect the text, e.g. when the text is forced to surround a drawing shape. [Open05]

- **Text content suppliers** include text tables, text fields, graphic objects,... [Open05][27]
- The structuring and styling of the text is done by the **objects for styling and numbering**. They provide style families for paragraphs, characters, pages, numbering and so on.[28] [Open05]

Figure 8 tries to provide an overview of the text document model.

5.2 The first Snippet

The first very simple snippet wants to illustrate some basic facts about OpenOffice.org Writer automation and the two ways our Snippets will be executed.

The first way is to directly write an `*.rex` file, which can be executed from the dos-prompt on Windows or the shell on Linux (a double-click is fine too).[29]

When looking in your BSF4frexx directory you will find a file called `testOOo.rex`, letting it run, will start OpenOffice.org and open a new Writer document. As it is quite good commented, there is no need to talk about it here in detail.

The most important lines in `testOOo.rex` are:

```
<snip>
Author:  Rony G. Flatscher
<snip>
componentLoader = UNO.createDesktop()~XDesktop~XComponentLoader
```

The Desktop is created and using the interfaces `XDesktop` and `XComponentLoader` the variable `componentLoader` is defined. Remember the twiddle sign from the introduction into Rexx, the interfaces can be seen as messages sent to `UNO.createDesktop()`[30].

```
writerComponent =  componentLoader~-
    loadComponentFromURL("private:factory/swriter","_blank",
0,.UNO~noProps)
```

[27] see `InsertAnnotation.rex` for an example
[28] see `AddPageNumbering.rex` for an example
[29] These scripts will have the comment /* run from command-line */ at the beginning.
[30] `UNO.createDesktop` is a routine defined in `UNO.CLS`. Giving it no arguments between the two brackets it will return a default `xContext` which is the component context [Open05]

Now the variable `writerComponent` is defined and a message to the `componentLoader` is sent to open a new document.

To get the text from the document :

```
text = writerComponent~XTextDocument~getText
```

We know `XTextDocument` and its function `getText` already from figure 8. Finally the desired text is written to the document with:

```
text~setString("Hello OpenOffice.org/StarOffice, this is" -
    "ooRexx speaking! ("date("S") time()")")[31]
```

Of course, `UNO.CLS` is included using:

```
::requires UNO.CLS
```

The second way for running our Snippets is from inside OpenOffice.org. Opening the macro dialogue as described in 4.5 and shown in figure 7 we can create a new snippet.[32]

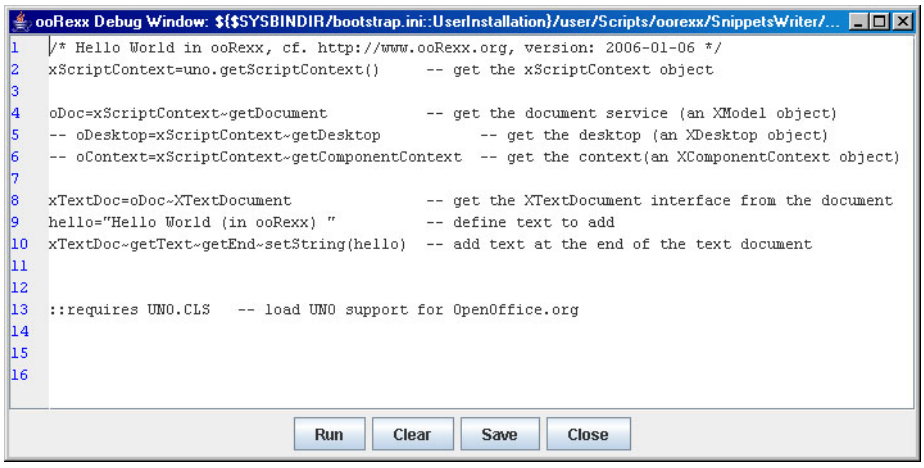

Figure 9: The macro editor with the ready-to-go code

After doing this, the macro editor presents us a ready-to-go code to insert the string „Hello World (in ooRexx)“ to the current document.

```
/*Hello World in ooRexx, cf. http://www.ooRexx.org, version: 2006-01-06*/
xScriptContext=uno.getScriptContext()
```

Getting the `xScriptContext` object is required for macros, so we have to do this in every macro.

[31] The `date()` and `time()` functions are Rexx specific. For more information you can look them up in [OoRx05].

[32] These scripts will have the comment /* Macro */ at the beginning.

```
oDoc=xScriptContext~getDocument
```

The `oDoc` can be compared to the `writerComponent` from above and is the the service manager from the document model, we learned about before.

The rest of the procedure is much the same as before and should be self-explaining.

```
xTextDoc=oDoc~XTextDocument
hello="Hello World (in ooRexx)"
xTextDoc~getText~getEnd~setString(hello)
::requires UNO.CLS
```

6 Examples

6.1 AddingFormattedText.rex

In this example we are going to add text into an opened Writer document using the macro editor. We want it to be bold, double underlined, Times New Roman and of size 15. Figure 10 shows the complete program code.

```
/* AddingFormattedText.rex */
/* Macro */
/* Text shall be underlined, Times New Roman, size 15 and bold */

/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()

oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* get the TextCursor from the interface's Text*/
xTextCursor=xTextDoc~getText~createTextCursor

/*setting the properties of the cursor using the XPropertySet interface*/
CursorProperties=xTextCursor~XPropertySet
CursorProperties~setPropertyValue("CharFontName", "Times New Roman")
CursorProperties~setPropertyValue("CharWeight",box("float", -
     bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))
     --Value is 150.0
CursorProperties~setPropertyValue("CharHeight", box("float", "15"))
CursorProperties~setPropertyValue("CharUnderline", box("short", -
     bsf.getConstant("com.sun.star.awt.FontUnderline", "DOUBLE")))
     --Value is 2

/* set text at the end of the document */
xTextDoc~getText~getEnd~setString("This is OORexx formatted text!")

::requires UNO.CLS
```

Figure 10: AddingFormattedText.rex

The following codeparts are cutouts of the code in Figure 10.

```
/* AddingFormattedText.rex */
/* Macro */
/* Text shall be underlined, Times New Roman, size 15 and bold */
/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument
```

We get the `XModel` and the `XTextDocument` interface. Actually we don't have to care much about it, as these lines are added automatically by the Macro Editor.

```
/* get the TextCursor from the interface's Text*/
xTextCursor=xTextDoc~getText~createTextCursor
```

We need the `xTextCursor` to set the properties of the text to add.

```
/*setting the properties of the cursor using the XPropertySet interface*/
CursorProperties=xTextCursor~XPropertySet
CursorProperties~setPropertyValue("CharFontName", "Times New Roman")
```

The `CharFontName` has to be a string value. [Open05]

```
CursorProperties~setPropertyValue("CharWeight", box("float", -
    bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))
    --Value is 150.0
```

When we want to change the `CharWeight`[33] we have to use two methods from `BSF.CLS`. The `bsf.getConstant()` method retrieves a value for "`BOLD`", which is in this case `150.0`. The `box` method converts it into a float variable, as `CharWeight` requests float values. Using the value `150.0` we could also set `CharWeight` directly as it is done for `CharHeight` with the value `15`.

```
CursorProperties~setPropertyValue("CharHeight", box("float", "15"))
CursorProperties~setPropertyValue("CharUnderline", box("short", -
    bsf.getConstant("com.sun.star.awt.FontUnderline", "DOUBLE")))
    --Value is 2
```

Here, again the two `BSF.CLS` methods are used. Other possible values instead of "`DOUBLE`" would be "`SINGLE`" or "`DOTTED`". [Open05]

```
/* set text at the end of the document */
xTextDoc~getText~getEnd~setString("This is OORexx formatted text!")

::requires UNO.CLS
```

Finally, the text is inserted.

[33] as done in [Aham05]

6.2 ChangingFormattedText.rex

In this example, which is shown in Figure 11, we are going to change the formatting of an existing text. After selecting it, we can change its font by running the script.

```
/* ChangingFormattedText.rex */
/* Macro */

/* get the script context, the Desktop and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop

/* get the xModel from the current Component and the xViewCursor */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
xViewCursorSupplier = xController~XTextViewCursorSupplier
xViewCursor = xViewCursorSupplier~getViewCursor()

/* set the properties for the xViewCursor */
xCursorPropertySet = xViewCursor~XPropertySet
xCursorPropertySet~setPropertyValue("CharFontName", "Courier New")

::requires UNO.CLS
```

Figure 11: ChangingFormattedText.rex

```
/* ChangingFormattedText.rex */
/* Macro */
/* get the script context, the Desktop and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop
```

In this example we get the Desktop to get the `xModel` from the current Component.

```
/* get the xModel from the current Component and the xViewCursor */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
xViewCursorSupplier = xController~XTextViewCursorSupplier
xViewCursor = xViewCursorSupplier~getViewCursor()
```

We now need the `xViewCursor`, which defines a range of text. [Open05]

The property setting is like we did it for the `xTextCursor` in `AddingFormattedText.rex`.

```
/* set the properties for the xViewCursor */
xCursorPropertySet = xViewCursor~XPropertySet
xCursorPropertySet~setPropertyValue("CharFontName", "Courier New")
::requires UNO.CLS
```

Now, when selecting a text and running the script the font will change to "Courier New".

6.3 DocumentBackground.rex

Now we will change the background color of a given document using the property value „BackColor“ as shown in Figure 12.

```
/* DocumentBackground.rex */
/* Macro */

/* get the script context and the Document */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument

/* The xFamiliesSupplier includes the PageStyles */
xFamiliesSupplier=oDoc~XStyleFamiliesSupplier
xStyle=xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~-
     XNameContainer

/* the background is part of the PageStyle */
xBackground = xStyle~getByName("Standard")~xPropertySet
/* set the background color */
xBackground~setPropertyValue("BackColor", box("int", "00ff00"x ~c2d))

::requires UNO.CLS
```

Figure 12: DocumentBackground.rex

```
/* DocumentBackground.rex */
/* Macro */
/* get the script context and the Document */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument

/* The xFamiliesSupplier includes the PageStyles */
xFamiliesSupplier=oDoc~XStyleFamiliesSupplier
xStyle=xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~-
    XNameContainer
```

The `XStyleFamiliesSupplier` can be found in figure 8 and is needed to access the `PageStyles`.

```
/* the background is part of the PageStyle */
xBackground = xStyle~getByName("Standard")~xPropertySet
/* set the background color */
xBackground~setPropertyValue("BackColor", box("int", "00ff00"x ~c2d))

::requires UNO.CLS
```

The background is modified using `xPropertySet` and setting the value for the colour. We can either insert an integer value directly or, convert the colour value to an integer using the box function and `"00ff00"x ~c2d` which is easi-

er, because we can use the RGB[34] model. [Aham05] `"00ff00"` sets the background colour to a bright green.

[34] Red, Green, Blue. Colour model, where each two digits represent the amount of each colour.

6.4 AddPageNumbering.rex

Using a Macro we want to add something like „Page 3 of 10“ into the header of each page of a document. Figure 13 demonstrates the difference between the total number of pages and the actual page number.

```
/* AddPageNumbering.rex */
/* Macro */

/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* access the PageStyles using XMultiServiceFactory */
xServiceManager=oDoc~XMultiServiceFactory
xPageStyle=xServiceManager~createInstance("com.sun.star.style.PageStyle")
xFamiliesSupplier = xTextDoc~XStyleFamiliesSupplier
xStyle=xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~-
     XNameContainer

/* set the Header up */
xHeader=xStyle~getByName("Standard")
HeaderProperty=xHeader~XPropertySet
HeaderProperty~setPropertyValue("HeaderIsOn", box("boolean", .true))
headerText=HeaderProperty~getPropertyValue("HeaderText")~XText

/* Creating the total number of pages */
pageCount=xServiceManager~-
     createInstance("com.sun.star.text.TextField.PageCount")
pageCountTC=pageCount~XTextContent()
pageCountPS=pageCount~XPropertySet()
pageCountPS~setPropertyValue("NumberingType", box("Short",-
     bsf.getConstant("com.sun.star.style.NumberingType", "ARABIC")))

/* Creating the actual page number */
pageNumber=xServiceManager~-
     createInstance("com.sun.star.text.TextField.PageNumber")
pageNumberTC=pageNumber~XTextContent()
pageNumberPS=pageNumber~XPropertySet()
pageNumberPS~setPropertyValue("NumberingType", box("Short", -
     bsf.getConstant("com.sun.star.style.NumberingType", "ARABIC")))
pageNumberPS~setPropertyValue("SubType", box("Short", "1"))

/* Insert text in the header */
headerText~setString("Page ")
headerText~insertTextContent(headerText~getEnd, pageNumberTC, .false)
headerText~getEnd~setString(" of ")
headerText~insertTextContent(headerText~getEnd, pageCountTC, .false)

::requires UNO.CLS
```

Figure 13: AddPageNumbering.rex

```
/* AddPageNumbering.rex */
/* Macro */

/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* access the PageStyles using XMultiServiceFactory */
xServiceManager=oDoc~XMultiServiceFactory
xPageStyle=xServiceManager~createInstance("com.sun.star.style.PageStyle")
xFamiliesSupplier = xTextDoc~XStyleFamiliesSupplier
xStyle=xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~-
    XNameContainer
```

For the header we need again the `PageStyles`, which we get from `XStyleFamiliesSupplier`.

```
/* set the Header up */
xHeader=xStyle~getByName("Standard")
HeaderProperty=xHeader~XPropertySet
HeaderProperty~setPropertyValue("HeaderIsOn", box("boolean", .true))
headerText=HeaderProperty~getPropertyValue("HeaderText")~XText
```

We retrieve the `headerText`, where we are going to insert text later on.

```
/* Creating the total number of pages */
pageCount=xServiceManager~-
    createInstance("com.sun.star.text.TextField.PageCount")
pageCountTC=pageCount~XTextContent()
pageCountPS=pageCount~XPropertySet()
```

The text content is stored in `pageCountTC` and its properties are stored in `pageCountPS`.

```
pageCountPS~setPropertyValue("NumberingType", box("Short", -
    bsf.getConstant("com.sun.star.style.NumberingType", "ARABIC")))
```

The `NumberingType` is defined as `"ARABIC"`, other possible types are `"CHARS_UPPER_LETTER"` or `"CHARS_LOWER_LETTER"`[35].

```
/* Creating the actual page number */
pageNumber=xServiceManager~-
    createInstance("com.sun.star.text.TextField.PageNumber")
pageNumberTC=pageNumber~XTextContent()
pageNumberPS=pageNumber~XPropertySet()
pageNumberPS~setPropertyValue("NumberingType", box("Short", -
    bsf.getConstant("com.sun.star.style.NumberingType", "ARABIC")))
```

We do the same procedure for the current page number.

```
pageNumberPS~setPropertyValue("SubType", box("Short", "1"))
```

To have the current page number also on the first page I found out, that it is necessary to put this statement.

[35] see http://api.openoffice.org/docs/common/ref/com/sun/star/style/NumberingType.html for a complete list.

```
/* Insert text in the header */
headerText~setString("Page ")
headerText~insertTextContent(headerText~getEnd,pageNumberTC, .false)
headerText~getEnd~setString(" of ")
headerText~insertTextContent(headerText~getEnd,pageCountTC, .false)
```

We now insert the text content of `pageNumberTC` and `pageCountTC` in the header.

The `insertTextContent` command defines where the content has to be inserted, and if existing content should be replaced or not (using `.false` or `.true`). [Star01]

```
::requires UNO.CLS
```

6.5 StoreAnyAsPDF.rex

In this example (as shown in Figure 14) we are going to open an existing `*.odt` file, convert it to a `*.pdf` file and close the Writer again. This script has to be started from the command line. Parts of this example were taken from [Aham05]. Look there for an example, which uses a file open dialog.

```
/* StoreAnyAsPDF.rex */
/* run from command line */
/* parts of this script are taken from A. Ahammer */

/* connect to server and get the service manager */
xContext = UNO.connect()
XMcf = xContext~getServiceManager

/* get the desktop and XComponentLoader */
xDesktop = UNO.createDesktop(xContext)
xComponentLoader = xDesktop~XDesktop~XComponentLoader

/* loading an existing document */
loadprops = bsf.createArray(.UNO~propertyValue, 1)
loadprops[1] = .UNO~PropertyValue~new
loadprops[1]~Name  = "Visible"
loadprops[1]~Value = box("boolean", .true)
xWriterComponent = xComponentLoader~-
     loadComponentFromURL("file:///C:/text.odt", "_blank", 0, loadprops)

/* storing the document as pdf */
xStorable = xWriterComponent~XStorable
storeprops = bsf.createArray(.UNO~propertyValue, 2)
storeprops[1] = .UNO~PropertyValue~new
storeprops[1]~Name  = "FilterName"
storeprops[1]~Value = "writer_pdf_Export"
storeprops[2] = .UNO~PropertyValue~new
storeprops[2]~Name  = "CompressMode"
storeprops[2]~Value = 2
xStorable~storeToUrl("file:///C:/text.pdf", storeprops)

/* close the Writer */
xWriterComponent~dispose()

::requires UNO.CLS
```

Figure 14: StoreAnyAsPDF.rex

```
/* StoreAnyAsPDF.rex */
/* run from command line */
/* parts of this script are taken from A. Ahammer */

/* connect to server and get the service manager */
xContext = UNO.connect()
XMcf = xContext~getServiceManager
```

We connect to the UNO server and request a service manager.

```
/* get the desktop and XComponentLoader */
xDesktop = UNO.createDesktop(xContext)
xComponentLoader = xDesktop~XDesktop~XComponentLoader
```

The component loader is necessary as it allows us to load documents from URLs.

```
/* loading an existing document */
loadprops = bsf.createArray(.UNO~propertyValue, 1)
loadprops[1] = .UNO~PropertyValue~new
loadprops[1]~Name  = "Visible"
loadprops[1]~Value = box("boolean", .true)
```

We create an array which contains the needed values for the loading. With the "Visible" tag set the document will be shown to us. Other possible entries for loadprops[1]~Name would be "AsTemplate", "Read-only" and "Hidden" where we would not see the Writer window. [Open05]

```
xWriterComponent = xComponentLoader~-
    loadComponentFromURL("file:///C:/text.odt", "_blank", 0, loadprops)
```

Using loadComponentFromURL we want to open the file text.odt. As OpenOffice.org needs an URL on we have to define its location in URL style. [Aham05]

On Windows machines c:\text.odt would convert to file:///C:/text.odt. On Linux machines /tmp/text.odt would convert to file:///tmp/text.odt.

```
/* storing the document as pdf */
xStorable = xWriterComponent~XStorable
```

To store the document we need the XStorable interface and we have to define an array with the needed values again.

```
storeprops = bsf.createArray(.UNO~propertyValue, 2)
storeprops[1] = .UNO~PropertyValue~new
storeprops[1]~Name  = "FilterName"
storeprops[1]~Value = "writer_pdf_Export"
storeprops[2] = .UNO~PropertyValue~new
storeprops[2]~Name  = "CompressMode"
storeprops[2]~Value = 2
xStorable~storeToUrl("file:///C:/text.pdf", storeprops)
```

With storeToUrl() we write the file to its location. [Aham05]

```
/* close the Writer */
xWriterComponent~dispose()
::requires UNO.CLS
```

xWriterComponent~dispose() disposes the Writer. [Open05]

6.6 StoreActualAsPDF.rex

Now, with the knowledge of the example above (Figure 14), we will look at a macro, which stores the actual opened file as `*.pdf`. Figure 15 shows the complete code.

```
/* StoreActualAsPDF.rex */
/* Macro */
/* How to store an opened file as *.pdf */

/* get the script context */
xScriptContext=uno.getScriptContext()

/* get the document and the component context */
oDoc=xScriptContext~getDocument
oContext=xScriptContext~getComponentContext

/* defining the storing properties */
xStorable = oDoc~XStorable
storeprops = bsf.createArray(.UNO~propertyValue, 2)
storeprops[1] = .UNO~PropertyValue~new
storeprops[1]~Name  = "FilterName"
storeprops[1]~Value = "writer_pdf_Export"
storeprops[2] = .UNO~PropertyValue~new
storeprops[2]~Name  = "CompressMode"
storeprops[2]~Value = 2

/* retrieve url from document */
url=oDoc~getURL()

/* if document has been saved to an url */
if (url<>"") then do
   parse var url url "." .
   url= url || ".pdf"
   xStorable~storeToUrl(url, storeprops)
end

/* if document has not been saved */
else do
   .bsf.dialog~messagebox("File has to be saved first.")
end

::requires UNO.CLS
```

Figure 15: StoreActualAsPDF.rex


```
/* StoreActualAsPDF.rex */
/* Macro */
/* How to store an opened file as *.pdf */

/* get the script context */
xScriptContext=uno.getScriptContext()

/* get the document and the component context */
oDoc=xScriptContext~getDocument
oContext=xScriptContext~getComponentContext
```

We start again by defining an array with the necessary storing properties.

```
/* defining the storing properties */
xStorable = oDoc~XStorable
storeprops = bsf.createArray(.UNO~propertyValue, 2)
storeprops[1] = .UNO~PropertyValue~new
storeprops[1]~Name  = "FilterName"
storeprops[1]~Value = "writer_pdf_Export"
storeprops[2] = .UNO~PropertyValue~new
storeprops[2]~Name  = "CompressMode"
storeprops[2]~Value = 2

/* retrieve url from document */
url=oDoc~getURL()
```

To get the URL from the document we use the `getURL()` function. [Open05]

If the variable `url` contains more than nothing (in our case more than nothing is the URL), the `*.pdf` file will be written to the same directory as the source file is in.

```
/* if document has been saved to an url */
if (url<>"") then do
  parse var url url "." .
  url= url || ".pdf"
  xStorable~storeToUrl(url, storeprops)
end
```

If the document has not been saved first, the macro would not save anything as `*.pdf`, so we show a message box, that informs about the fact, that the document has to be saved.

```
/* if document has not been saved */
else do
  .bsf.dialog~messagebox("File has to be saved first.")
end

::requires UNO.CLS
```

6.7 InsertASCII.rex

In this example (see Figure 16) it is shown, how to insert very easyly line breaks and tabulator tabs.

```
/* InsertASCII.rex */
/* Macro */
/* Insert ASCII using the decimal value */

/* get xScriptContext the document service and the text document */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* carriage return */
cr="13"~d2c
/* tabulator */
tab="09"~d2c

/* insert cr */
xTextDoc~getText~getEnd~setString(cr)

::requires UNO.CLS
```

Figure 16: InsertASCII.rex

Looking at an ASCII table, you will find out, that each character has got its decimal counterpart. For example a is 97 and N is 98[36]. But also control characters have got it's unique number, which we are going to use now.

```
/* InsertASCII.rex */
/* Macro */
/* Insert ASCII using the decimal value */

/* get xScriptContext the document service and the text document */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* carriage return */
cr="13"~d2c

/* tabulator */
tab="11"~d2c
```

The carriage return, which is a new line in our context, has got number 13 in the table. Using `~d2c` we can convert the decimal value to a character. `d2c` is a rexx function, that converts a **d**ecimal to **c**haracter. [OoRx05]

[36] Go to http://www.lookuptables.com/ to view a complete table of ASCII codes.

The tabulator has got number 09 and is converted the same way.

```
/* insert cr */
xTextDoc~getText~getEnd~setString(cr)

::requires UNO.CLS
```

Inserting the carriage returns character representation we will end up in a new line.

In [Aham05] you will find an example, which contains this line:

```
xText~insertControlCharacter(xTextCursor, -
    bsf.getConstant("com.sun.star.text.ControlCharacter", -
    "PARAGRAPH_BREAK"), .false)
```

Replacing “PARAGRAPH_BREAK”[37] with “LINE_BREAK”[38] is another way to get into a new line. [Open05]

[37] A paragraph break always is performed, when hitting the return key.
[38] Hitting Ctrl+Return / Strg+Return will result in a line break.

6.8 InsertPageBreak.rex

This Macro (Figure 17) uses the ascii character conversion from the code in Figure 16 to insert some text, makes a pagebreak and inserts some text again.

```
/* InsertPageBreak.rex */
/* Macro */

/* get the script context, the document service and the XTextDocument */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* create the cursor */
xTextCursor=xTextDoc~getText~createTextCursor()
xText=xTextDoc~getText()
xTextDoc~getText~getEnd~setString("This text is on page number 1.")

/* set cursor properties and define BreakType */
xCursorProps=xTextCursor~XPropertySet
xCursorProps~setPropertyValue("BreakType", -
     bsf.getConstant("com.sun.star.style.BreakType","PAGE_AFTER"))
xTextDoc~getText~getEnd~setString("13" ~d2c)

/* set text on page 2 */
xTextDoc~getText~getEnd~setString("This text is on page number 2.")

::requires UNO.CLS
```

Figure 17: InsertPageBreak.rex

```
/* InsertPageBreak.rex */
/* Macro */

/* get the script context, the document service and the XTextDocument */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument
```

We need to create the cursor, because the pagebreak property belongs to the `TextCursor`.

```
/* create the cursor */
xTextCursor=xTextDoc~getText~createTextCursor()
xText=xTextDoc~getText()
xTextDoc~getText~getEnd~setString("This text is on page number 1.")
```

Now we retrieve the cursor properties and set the `BreakType` value to `"PAGE_AFTER"`. [Open05] Followed by a carriage return, which will bring us onto the next page.

```
/* set cursor properties and define BreakType */
xCursorProps=xTextCursor~XPropertySet
xCursorProps~setPropertyValue("BreakType", -
    bsf.getConstant("com.sun.star.style.BreakType", "PAGE_AFTER"))
xTextDoc~getText~getEnd~setString("13"x~d2c)
```

Finally another text is inserted on the new page.

```
/* set text on page 2 */
xTextDoc~getText~getEnd~setString("This text is on page number 2.")
::requires UNO.CLS
```

It is worth mentioning, that after inserting the pagebreak it is not possible to insert a paragraph break on that page anymore, while line breaks stay available.

6.9 TableOfContents.rex

We are going to insert a table of contents using the different titles and subtitles of a document. Figure 18 shows how to set it up correctly, again we have to differentiate between the property set and the text content, as we did in Figure 13.

```
/* TableOfContents.rex */
/* Macro */
/* inserts a table of contents in a given document */

/* get the script context, the document service and the XTextDocument */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument

/* retrieve XMultiServiceFactory to get the ContentIndex */
xServiceManager=oDoc~XMultiServiceFactory
toc=xServiceManager~createInstance("com.sun.star.text.ContentIndex")

/* access the properties: level 10, use headings */
tocPS = toc~XPropertySet
tocPS~setPropertyValue("Level", box("Short", "10"))
tocPS~setPropertyValue("CreateFromOutline", box("boolean" ,.true))

/* get the text content of toc and insert it */
tocTC = toc~XTextContent()
xText=xTextDoc~getText()
xCursor=xTextDoc~getText~createTextCursor()
xText~insertTextContent(xCursor, tocTC, .false)

/* retrieve XDocumentIndex and update it */
xDocIndex=toc~XDocumentIndex
xDocIndex~update()

::requires UNO.CLS
```

Figure 18: TableOfContents.rex


```
/* TableOfContents.rex */
/* Macro */
/* inserts a table of contents in a given document */

/* get the script context, the document service and the XTextDocument */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument
```


The `ContentIndex` is part of the `XmultiServiceFactory`, so we have to get it and create an instance of `"com.sun.star.text.ContentIndex"`.

```
/* retrieve XMultiServiceFactory to get the ContentIndex */
xServiceManager=oDoc~XMultiServiceFactory
toc=xServiceManager~createInstance("com.sun.star.text.ContentIndex")
```

We access the properties and set it to level 10 and to use the outline. These settings can be found in the normal dialogue window, you can use to create an index, as well. [Open02]

```
/* access the properties: level 10, use headings */
tocPS = toc~XPropertySet
tocPS~setPropertyValue("Level", box("Short", "10"))
tocPS~setPropertyValue("CreateFromOutline", box("boolean" ,.true))
```

We retrieve the text content from the table of contents object and insert it.

```
/* get the text content of toc and insert it */
tocTC = toc~XTextContent()
xText=xTextDoc~getText()
xCursor=xTextDoc~getText~createTextCursor()
xText~insertTextContent(xCursor, tocTC, .false)
```

To actually display the right text in the table of contents, we have to update it once.

```
/* retrieve XDocumentIndex and update it */
xDocIndex=toc~XDocumentIndex
xDocIndex~update()

::requires UNO.CLS
```

6.10 MailMerge.rex

The following stand-alone script (Figure 19) will run a mail merge. Using the address data from the file `addresses.ods` and the letter file `letter.odt` it will create a new text document with one address and the text from the letter on each page.

```
/* MailMerge.rex */
/* run from command line */
/* runs a MailMerge using an existing *.ods file */

/* get the desktop and a component loader */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader

/* open Calc and get first sheet in spreadsheet */
url = "file:///c:/addresses.ods"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, -
     "_blank", 0, .UNO~noProps)
xSheet=xCalcComponent~XSpreadSheetDocument~getSheets~-
     XIndexAccess~getByIndex(0)~XSpreadSheet
/* open a blank document in Writer */
url= "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, -
     "_blank", 0, .UNO~noProps)
xText=xWriterComponent~XTextDocument~getText()
/* start at line 1 in Calc */
line = 0
/* do this until empty cell text is found */
do while xSheet~getCellByPosition(0,line)~getFormula() <> ""
   /* read all cell texts */
   surname = xSheet~getCellByPosition(0,line)~getFormula()
   familyname = xSheet~getCellByPosition(1,line)~getFormula()
   address = xSheet~getCellByPosition(2,line)~getFormula()
   zip = xSheet~getCellByPosition(3,line)~getFormula()
   city = xSheet~getCellByPosition(4,line)~getFormula()
   /* insert text in Writer */
   xText~getEnd~setString(surname || " " || familyname)
   call newline 1
   xText~getEnd~setString(address)
   call newline 1
   xText~getEnd~setString(zip || " " || city)
   call newline 5
   xText~getEnd~setString("Dear " || surname ||"!")
   call newline 2
   /* insert the letter */
   xTextCursor = xText~getText~createTextCursor
   insertprops = bsf.createArray(.UNO~propertyValue, 0)
   xTextCursor~gotoEnd(.false)
   xTextCursor~XDocumentInsertable~-
       insertDocumentFromURL("file:///C:/letter.odt", insertprops)
   /* perform a pagebreak */
   xCursorProps=xTextCursor~XPropertySet
   xCursorProps~setPropertyValue("BreakType", -
       bsf.getConstant("com.sun.star.style.BreakType","PAGE_AFTER"))
   call newline 1
   line= line + 1
end

EXIT 0
```

Figure 19: MailMerge.rex (part 1)

```
/* function for inserting more than one carriage returns */
newline:
use arg count
do count
   xText~getEnd~setString("13" ~d2c)
end
return

::requires UNO.CLS
```

Figure 20: MailMerge.rex (part 2)

```
/* MailMerge.rex */
/* run from command line */
/* runs a MailMerge using an existing *.ods file */

/* get the desktop and a component loader */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader
```

We first open Calc and point it to the first spreadsheet. We have to use `0` as the index, as it starts counting the lines and columns at `0` and not, as we are used to at `1`. [Aham05]

```
/* open Calc and get first sheet in spreadsheet */
url = "file:///c:/addresses.ods"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, -
    "_blank", 0, .UNO~noProps)
xSheet=xCalcComponent~XSpreadSheetDocument~getSheets~-
    XIndexAccess~getByIndex(0)~XSpreadSheet
```

For the output, we open an OpenOffice.org Writer and an empty document.

```
/* open a blank document in Writer */
url= "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, -
    "_blank", 0, .UNO~noProps)
xText=xWriterComponent~XTextDocument~getText()
```

Again, we have to be aware, that line 1 is actually line 0 in our code.

```
/* start at line 1 in Calc */
line = 0
```

The file `addresses.ods` is read. We read 5 columns per line until we arrive at an empty cell.

Using `getFormula()` we can access the formula of a cell. [Open05] This works also for regular strings.

```
/* do this until empty cell text is found */
do while xSheet~getCellByPosition(0,line)~getFormula() <> ""
  /* read all cell texts */
  surname = xSheet~getCellByPosition(0,line)~getFormula()
  familyname = xSheet~getCellByPosition(1,line)~getFormula()
  address = xSheet~getCellByPosition(2,line)~getFormula()
  zip = xSheet~getCellByPosition(3,line)~getFormula()
  city = xSheet~getCellByPosition(4,line)~getFormula()
```

The text is inserted. The carriage returns are created using a function, which is described below.

```
  /* insert text in Writer */
  xText~getEnd~setString(surname || " " || familyname)
  call newline 1
  xText~getEnd~setString(address)
  call newline 1
  xText~getEnd~setString(zip || " " || city)
  call newline 5
```

Using "||" concatenates two strings. We know this operator from chapter 2.2.3.

```
  xText~getEnd~setString("Dear " || surname ||"!")
  call newline 2
```

The prefabricated letter file `letter.odt` is inserted using `insertDocument-fromURL()`. [Open05]

```
  /* insert the letter */
  xTextCursor = xText~getText~createTextCursor
  insertprops = bsf.createArray(.UNO~propertyValue, 0)
  xTextCursor~gotoEnd(.false)
  xTextCursor~XDocumentInsertable~-
      insertDocumentFromURL("file:///C:/letter.odt", insertprops)
```

We do a pagebreak to separate the letters from each other, so every letter is written on its own page.

```
  /* perform a pagebreak */
  xCursorProps=xTextCursor~XPropertySet
  xCursorProps~setPropertyValue("BreakType", -
      bsf.getConstant("com.sun.star.style.BreakType", "PAGE_AFTER"))
  call newline 1
  line= line + 1
end
EXIT 0
```

The loop for reading the cell texts ends at the `end` command.

This function uses the argument provided in the `call` command. According to the `call` argument the number of carriage returns is inserted.[39]

[39] see [OoRx05] for an detailed explanation to the CALL command

```
/* function for inserting more than one carriage returns */
newline:
use arg count
do count
  xText~getEnd~setString("13" ~d2c)
end
return

::requires UNO.CLS
```

6.11 InsertAnnotation.rex

The snippet shown in Figure 21 inserts an annotation from the author „ooRexx“ and with the content „I was here.“.

```
/* InsertAnnotation.rex */
/* Macro */

/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument
xText=xTextDoc~getText()

/* get the XMultiServiceFactory needed for textfields */
xServiceManager=oDoc~XMultiServiceFactory

/* create an annotation */
annotation=xServiceManager~-
     createInstance("com.sun.star.text.TextField.Annotation")
annotationTC=annotation~XTextContent()

/* set author and content of the annotation */
annotationPS=annotation~XPropertySet()
annotationPS~setPropertyValue("Author", "ooRexx")
annotationPS~setPropertyValue("Content", "I was here.")

/* insert annotation at cursor position */
xCursor=xTextDoc~getText~createTextCursor()
xText~insertTextContent(xCursor, annotationTC, .false)

::requires UNO.CLS
```

Figure 21: InsertAnnotation.rex

```
/* InsertAnnotation.rex */
/* Macro */

/* get the script context, the XModel and the XTextDocument interface */
xScriptContext=uno.getScriptContext()
oDoc=xScriptContext~getDocument
xTextDoc=oDoc~XTextDocument
xText=xTextDoc~getText()
```

As we already know from the `AddPageNumbering.rex` example we need the `XmultiServiceFactory`, which provides different textfields.

```
/* get the XMultiServiceFactory needed for textfields */
xServiceManager=oDoc~XMultiServiceFactory

/* create an annotation */
annotation=xServiceManager~-
    createInstance("com.sun.star.text.TextField.Annotation")
annotationTC=annotation~XTextContent()
```

The annotation is created very much like the `PageNumber` and `PageCount` fields. We distinguish between the text content (`annotationTC`) and the properties (`annotationPS`).

The author and the content of the annotation is set:

```
/* set author and content of the annotation */
annotationPS=annotation~XPropertySet()
annotationPS~setPropertyValue("Author", "ooRexx")
annotationPS~setPropertyValue("Content", "I was here.")
```

Finally, the annotation is inserted at the cursor.

```
/* insert annotation at cursor position */
xCursor=xTextDoc~getText~createTextCursor()
xText~insertTextContent(xCursor, annotationTC, .false)

::requires UNO.CLS
```

Executing this script will insert an annotation, which can be detected because of a little yellow box in the document.

6.12 HideAnnotations.rex

Now, that we've inserted an annotation (look at Figure 21), we will look at a script (Figure 22), that hides existing annotations from us.

```
/* HideAnnotations.rex */
/* Macro */
/* hide existing annotations */

/* get the script context and the desktop */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop

/* get the current component (the Writer) and the controller from the
xModel */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()

/* the XViewSettings is part of the xSelectionSupplier */
xSelectionSupplier = xController~XSelectionSupplier
xViewSettingsSupplier = xSelectionSupplier~XViewSettingsSupplier
xViewSettings=xViewSettingsSupplier~ViewSettings

/* get the property set for the current view and hide the annotations*/
xViewProperties = xViewSettings~XPropertySet
xViewProperties~setPropertyValue("ShowAnnotations", box("boolean",.false))

::requires UNO.CLS
```

Figure 22: HideAnnotations.rex

```
/* HideAnnotations.rex */
/* Macro */
/* hide existing annotations */

/* get the script context and the desktop */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop
```

As in `ChangingFormattedText.rex` we need the `currentController` from the Writer component again.

```
/* get the current component (the Writer) and the controller from the
xModel */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
```

We can derive the `XViewSettings` from the `XSelectionSupplier`.

```
/* the XViewSettings is part of the xSelectionSupplier */
xSelectionSupplier = xController~XSelectionSupplier
xViewSettingsSupplier = xSelectionSupplier~XViewSettingsSupplier
xViewSettings=xViewSettingsSupplier~ViewSettings
```

To alter the settings of the view, we have to access its `XpropertySet`.

```
/* get the property set for the current view and hide the annotations */
xViewProperties = xViewSettings~XPropertySet
xViewProperties~setPropertyValue("ShowAnnotations", box("boolean",.false))
```

Now setting the property value "`ShowAnnotations`" to `false` we hide the annotations.

```
::requires UNO.CLS
```

6.13 AlterZoom1.rex

In our last example, we are going to change the zoom of a current document. We use again the `ViewSettings` as we did in `HideAnnotations.rex`. Figure 23 lists the different zoom states.

```
/* AlterZoom.rex */
/* Macro */

/* get the script context and the desktop */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop

/* get current component (Writer) and its xViewSettings */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
xSelectionSupplier = xController~XSelectionSupplier
xViewSettingsSupplier = xSelectionSupplier~XViewSettingsSupplier
xViewSettings=xViewSettingsSupplier~ViewSettings
xViewProperties = xViewSettings~XPropertySet

/* show different zooms with a 1 sec pause */
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", "PAGE_WIDTH")))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", -
"ENTIRE_PAGE")))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", -
     "PAGE_WIDTH_EXACT")))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", "BY_VALUE")))
xViewProperties~setPropertyValue("ZoomValue", box("Short", 200))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", "BY_VALUE")))
xViewProperties~setPropertyValue("ZoomValue", box("Short", 10))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
     bsf.getConstant("com.sun.star.view.DocumentZoomType", "OPTIMAL")))

::requires UNO.CLS
```

Figure 23: AlterZoom.rex

```
/* AlterZoom.rex */
/* Macro */
/* get the script context and the desktop */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop
```

We know the following steps already from `HideAnnotations.rex` (Figure22).

```
/* get current component (Writer) and its xViewSettings */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
xSelectionSupplier = xController~XSelectionSupplier
xViewSettingsSupplier = xSelectionSupplier~XViewSettingsSupplier
xViewSettings=xViewSettingsSupplier~ViewSettings
xViewProperties = xViewSettings~XPropertySet
```

Now, the Zoom can be accessed using "`ZoomType`", which has different options:

```
/* show different zooms with a 1 sec pause */
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", "PAGE_WIDTH")))
```

Zoom is set to be as big as the page is wide, followed by a one second pause to enable the user to see the change.

```
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", "ENTIRE_PAGE")))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", -
    "PAGE_WIDTH_EXACT")))
call syssleep 1
```

To define our own zoom level we have to set the `DocumentZoomType` to "`BY_VALUE`" followed by a "`ZoomValue`" identifier. "`ZoomValue`" requires short values, so we have to use the box function.

```
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", "BY_VALUE")))
xViewProperties~setPropertyValue("ZoomValue", box("Short", 200))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", "BY_VALUE")))
xViewProperties~setPropertyValue("ZoomValue", box("Short", 10))
call syssleep 1
xViewProperties~setPropertyValue("ZoomType", box("Short", -
    bsf.getConstant("com.sun.star.view.DocumentZoomType", "OPTIMAL")))

::requires UNO.CLS
```

6.14 AlterZoom2.rex

An easier way to access the different DocumentZoomTypes is to use `bsf.wrapStaticFields` as is done in Figure 24.

```
/* AlterZoom2.rex */
/* Macro */
/* get the script context and the desktop */
xScriptContext=uno.getScriptContext()
xDesktop=xScriptContext~getDesktop

/* get current component (Writer) and its xViewSettings */
xComponent = xDesktop~getCurrentComponent()
xModel = xComponent~xModel
xController = xModel~getCurrentController()
xSelectionSupplier = xController~XSelectionSupplier
xViewSettingsSupplier = xSelectionSupplier~XViewSettingsSupplier
xViewSettings=xViewSettingsSupplier~ViewSettings
xViewProperties = xViewSettings~XPropertySet

/* access the static fields of DocumentZoomType */
c=bsf.wrapStaticFields("com.sun.star.view.DocumentZoomType")

/* show different zooms with a 1 sec pause */
xViewProperties~setPropertyValue("ZoomType", box("Short", c~page_width))
call syssleep 1

xViewProperties~setPropertyValue("ZoomType",  box("Short", c~entire_page))
call syssleep 1

xViewProperties~setPropertyValue("ZoomType",  box("Short", -
     c~page_width_exact))
call syssleep 1

xViewProperties~setPropertyValue("ZoomType",  box("Short", c~by_Value))
xViewProperties~setPropertyValue("ZoomValue", box("Short", 200))
call syssleep 1

xViewProperties~setPropertyValue("ZoomValue", box("Short", 10))
call syssleep 1

xViewProperties~setPropertyValue("ZoomType",  box("Short", c~optimal))

::requires UNO.CLS
```

Figure 24: AlterZoom2.rex

```
/* access the static fields of DocumentZoomType */
c=bsf.wrapStaticFields("com.sun.star.view.DocumentZoomType")
```

The static field of DocumentZoomType are accessed and stored in the variable c.

```
xViewProperties~setPropertyValue("ZoomType", box("Short", c~page_width))
/* <snip> */
xViewProperties~setPropertyValue("ZoomType",  box("Short", c~optimal))
```

The diferent ZoomTypes are sent as messages to the variable c.

7 Conclusion

Starting again with the research question „How can the OpenOffice.org Writer be automated using ooRexx?“ it now becomes clear, after having read this paper, that the automation of OpenOffice.org Writer is possible using ooRexx.

It is a good way for saving time on tasks, that have to be performed very often. With the ability of OpenOffice.org to be controlled using the Java programming language it offers a good possibility to be automated with a commonly used language.

Using the BSF4Rexx package enables ooRexx to control OpenOffice.org. As ooRexx is a script language and additionally very easy to understand the automation of OpenOffice.org even becomes more convenient.

The given Snippet examples can be seen as a little library for finding help on different tasks and give ideas on how to solve new problems.

With a little understanding of Java and Object-oriented programming it is possible to translate existing Java programmes to ooRexx and furthermore write new scripts in ooRexx.

The main difficulties during this work concerned the way, how to tell OpenOffice.org using ooRexx, what to do, which was very often trial and error. The Developer's Guide is a very good source for help with problem solving, but could be sometimes a bit more practical with additional examples on how to call different functions of OpenOffice.org.

When looking at the code snippets page[40] of OpenOffice.org one soon will find out that there is already a little community built around the automating abilibilties of OpenOffice.org which is still growing and will possibly become a place for everyone who is interested in this topic and wants to contribute or get further help.

[40] http://codesnippets.services.openoffice.org/ (Also the snippets from this paper are listed there)

8 References

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

[ApJa01] The Jakarta Project, http://jakarta.apache.org/, retrieved on 2006-05-30

[ApXa01] The Apache Xalan Project, http://xalan.apache.org/, retrieved on 2006-05-30

[Augu05] Walter Augustin, Examples for Open Office Automation with Scripting Languages, Bachelor Course Pager, 2005

[BSFR01] BSF4Rexx readme, http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/readmeBSF4Rexx.txt, last retrieved on 2006-06-20

[Burg05] Martin Burger, OpenOffice.org Automatisation with Object Rexx, Bachelor Course Paper, 2005

[Flat01] The Vienna Version of BSF4Rexx, Rony G. Flatscher, http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf, retrieved on 2006-05-31

[Flat02] UNO.CLS: An (Open) Object Rexx Module for Universal Network Objects, http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_UNO.pdf, retrieved on 2006-06-20

[IBMA01] IBM Alpha Works, http://www.alphaworks.ibm.com/about, retrieved on 2006-05-30

[IBMW01] Watson Research Center, http://www.watson.ibm.com/, retrieved on 2006-05-30

[Java01] About the Java Technology, http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html, last retrieved on 2006-06-20

[Java02] Download Java Platform, http://java.sun.com/javase/6/download.jsp, last retrieved on 2006-06-20

[Java03] Update: An Introduction to the Java EE 5 Platform, http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/, last retrieved on 2006-06-20

[Java04] Java ME Technology, http://developers.sun.com/techtopics/mobility/j2me/, last retrieved on 2006-06-20

[JBSF01] Jakarta BSF, FAQ, http://jakarta.apache.org/bsf/faq.html, retrieved on 2006-05-30

[JBSF02] Jakarta BSF, http://jakarta.apache.org/bsf/index.html, retrieved on 2006-05-30

[JBSF03] Jakarta BSF, Manual, http://jakarta.apache.org/bsf/manual.html, retrieved on 2006-05-30

[JBSu01] JavaBeans, http://java.sun.com/products/javabeans/, retrieved on 2006-05-30

[Niem02] Alexander Niemann, Objektorientierte Programmierung in Java, bhv, 2002

[Oasi01] OASIS Open Document Format for Office Applications FAQ, http://www.oasis-open.org/committees/office/faq.php, retrieved on 2006-05-10

[OoRx01] About Open Object Rexx, http://www.oorexx.org/index.html, retrieved on 2006-05-10

[OoRx02] Open Object Rexx Downloads, http://www.oorexx.org/download.html, last retrieved on 2006-06-26

[OoRx04] Object Rexx for Windows, Programming Guide, Rexx Language Association and others, 2004, http://www.oorexx.org/rexxpg.pdf, last retrieved on: 2006-06-26

[OoRx05] Open Object Rexx Reference, Version 3.0.0, Edition November 11, 2005, http://www.oorexx.org/rexxref.pdf, last retrieved on: 2006-06-26

[Open01] Native Language Confederation, http://projects.openoffice.org/native-lang.html, retrieved on 2006-05-10

[Open02] OpenOffice.org2.x User Guide, http://documentation.openoffice.org/manuals/OOo2.x/user_guide2_draft.pdf, retrieved on 2006-05-10

[Open03] OpenOffice.org 2 - Product Description, http://www.openoffice.org/product/index.html, retrieved on 2006-05-10

[Open04] About Us: OpenOffice.org – Historical background, http://about.openoffice.org/index.html#history, retrieved on 2006-05-10

[Open05] OpenOffice.org 2.0, Developer's Guide, 2005, http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf, retrieved on 2006-03-29

[Open06] Get OpenOffice.org!, http://download.openoffice.org/2.0.2/index.html, last retrieved on 2006-06-20

[Open07] CD-ROM Buyers' page, http://distribution.openoffice.org/cdrom/, last retrieved on 2006-06-26

[RxLa01] The Rexx Language Association - About RexxLA, http://rexxla.org/About_RexxLA/, retrieved on 2006-05-10

[Star01] StarOffice 8 Programmierhandbuch für Basic, http://docs.sun.com/app/docs/doc/819-1326/6n3mlokub?l=de&a=view, last retrieved on 2006-06-26

[Stey01] Ralph Steyer, Java 2 – Das Programmier-Handbuch, Markt+Technik, 2001

[Wiki01] Wikipedia: OpenOffice.org, http://en.wikipedia.org/wiki/Open_Office, retrieved on 2006-05-10

[Wiki02] Wikipedia: Rexx, http://en.wikipedia.org/wiki/REXX, retrieved on 2006-05-10