Vienna University of Economics and Business Administration

BACHELOR THESIS

English Title:

OpenOffice.org Calc Automation Using ooRexx

Author: Michael Hinz

Matriculation Number: h0253952

Field of Study: J033 526 Bakkalaureat Wirtschaftsinformatik

Course: 1236 Vertiefungskurs VI/Bakkalaureatsarbeit – Electronic Commerce

Text Language: English

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

I assure

that I have composed this bachelor thesis independently.

that I have only used quoted resources and no other unauthorised help.

that I have not submitted this thesis (whether at home nor abroad) to any judge for marking so far.

that this thesis is consistent with the marked thesis from the tutor.

Date _____ Signature _____

Table of Contents

List of Figures

11 Introduction

This chapter will give you an short overview about the content, the structure and the approach of this work.

1.1 Abstract

This paper gives an introduction to the OpenOffice.org architecture and explains how the OpenOffice.org Calc component can be automated by using the scripting language Open Object Rexx (ooRexx). This components are open sourced and can be downloaded free of charge from the internet.

The paper is divided into a theoretical and a practical part. In the theoretical part, the main components, ooRexx, OpenOffice.org and the Bean Scripting Framework for ooRexx, will be described and it explains how the single components can work together. At the end of this part you can find an short installation guide, which shows you how to retrieve and install the single components. The practical part provides some nutshell examples, that should demonstrate how the OpenOffice.org Calc component can be automated. The concluding part should give a short summary of the paper.

1.2 Research Question

How is it possible to build a bridge between OpenOffice.org and the scripting language ooRexx and how can the OpenOffice.org Calc be automated using ooRexx?

22 Open Object Rexx

2.1 History

This section gives an short overview about the development from REXX to ooRexx.

2.1.1 REXX

REXX[1](Restructured Extended Executor) was developed by Mike F. Cowlishow, an IBM employee, in 1979 to replace EXEC II, which was the batch language for IBM mainframes at that time. The idea was to create a „human centric language“, which is easy to learn and easy to work with. Over the years IBM implemented REXX to nearly all of its operating systems.
In 1996 REXX got standardized by the American National Standards Institute (ANSI). The standard was called 'ANSI "Programming Language - REXX", X3.274-1996'. [Flat06, p.1f]

2.1.2 ObjectRexx

ObjectRexx is an objectoriented approach to the scripting language Rexx. It was initiated in 1988 by a group of English IBM engineers[2]. Due to the request of the SHARE[3] SIG (special interest group), ObjectRexx is backwardly compatible with REXX, so that no existing REXX application had to be rewritten. After almost nine years of development and experimental designs, a commercial version called „Object REXX“ was released and implemented into OS/2. Also OS/2 independent versions of Object REXX for AIX and Windows were established. [Flat06, p.4]

---

1 The capitalized notation REXX refers to the IBM version
2 „Originally IBM's work on an object-oriented version of REXX was conducted in England under the lead of Simon Nash, then the project was transferred to the United States where finally a design and implementation under the lead of Rick McGuire succeeded.“[Flat06, p.4]
3 SHARE was the name of a special interest group that had a big influence on IBM

Since 2005 Object Rexx is distributed and developed under the responsibility of the non-profit-oriented SIG RexxLA[4], as an opensource software named „Open Object Rexx“ (ooRexx). [Flat06, p.5]

2.1.3 Open Object Rexx

Open Object Rexx is an open source scripting language, which can be characterised as follows:

- **An English-like language**:

  The idea of Rexx was to develop a „human centric language“, which uses common English words for instructions that have a similar semantic meaning in the English language.

  For Example Rexx uses words for instructions like SAY, PULL, IF...THEN...ELSE, DO...END, and EXIT. This approach makes it very easy to learn and to use the scripting language Rexx. [Oore06]

- **Fewer rules**:

  Rexx is not case sensitive, that means that Rexx instructions can be written in uppercase, lowercase or in mixed case. It is also possible to span an instruction over multiple lines or to write several instructions in one line, because there exists no line numbering. Due to that fact you can also skip entire lines or use multiple blanks in a line, without causing any trouble during running the program.
  Another feature is that variables can be named like build-in functions because the keywords are only reserved in context and the interpreter will use the right function. [Oore06]

- **Interpreted, not compiled**:

  Because Rexx is a scripting language it is interpreted and not compiled. [Oore06]

4 RexxLA - the Rexx Language Association

- **Built-in functions and methods**:

  Rexx offers build-in functions and methods that provide different operations and functionalities. These functions and methods are already implemented in Rexx. [Oore06]

- **Typeless variables**:

  You do not have to specify which type of variable is used, e.g. Strings or numbers, because a variable in Rexx can hold any kind of Object. [Oore06]

- **String handling**:

  Due to the fact that Rexx includes powerful functionalities for manipulating character strings, it is possible to read and separate characters, numbers, and mixed input. [Oore06]

- **Decimal Arithmetic**:

  Because humans base their arithmetic on decimal arithmetic, Rexx bases its arithmetic, different to other programming languages which base their arithmetic on binary arithmetic, on decimal arithmetic too. [Oore06]

- **Clear error messages and powerful debugging**:

  If an error is encountered while running a program, an error message with full and meaningful explanation is provided. Additionally Rexx provides a powerful debugging tool, the TRACE instruction. [Oore06]

2.2 Syntax

To understand the nutshell examples in this paper it is necessary to give you a quick introduction to the Object Rexx syntax.

- This code shows you two different possibilities to write a comment: [Flat06-1]

```
-- span several lines
/*
comment
comment
*/
```

```
-- span just one line
-- comment
```

- This example shows how to set variables and print them on the command shell: [Flat06-1]

```
-- without quotation marks every letter is capitalized
A = hello
b = "WoRld"
c = 15

SAY A b c

Output: HELLO WoRld 15
```

- The following code shows you several possibilities to realize a loop:
  Loops are realized within a block. A block starts with DO and ends with END: [Flat06-1]

```
DO
instructions
END
```

```
DO 3
SAY  "REXX!"
END
```

```
DO i = 1 TO 3
SAY  "REXX!"
END
```

```
i = 2
DO WHILE i < 3
SAY  "REXX"
i = i + 1
END
```

```
i = 3
DO UNTIL i > 1
SAY  "REXX"
i = i + 1
END
```

- The following example shows you hoe to realize an if-clause: [Flat06-1]

```
i = 10
IF i > 5 THEN DO
SAY "Hello World!"
END
```

- The following code shows how to set up a procedure, using the statement „::routine“, and how to invoke this procedur: [Flat06-2]

```
::routine name                  -- the name of the routine
use arg x, y                    -- the variables the routine needs
instructions
return z                        -- if the routine should return something
```

```
call name 2, 3

e.g. call syssleep 5                    -- the program will stop for 2 seconds
```

Note that „sysleep“ is a predefined function.

- requires – directive

The statement „::requires“ invokes another Rexx program. This is always the first statement, which is executed in a Rexx program, and all public routines of the other Rexx program are made available. In the examples below there is always one requirement: [Flat06-2]

```
::requires UNO.cls          -- get OOo support
```

To call an object’s method within Rexx you have to send a message to the object. Therefore the „Twiddle“ (~) is used. This will return whatever the method returns, and by using two Twiddles (~~) the object itself will be returned. The Twiddle is similar to the **.** in Java. [Flat05]

```
object1~method1         -- returns what the method1 returns
```

```
object1~~method1   -- returns object1
```

33 OpenOffice.org

3.1 History

In 1999 Sun Microsystems, Inc. took over a company named StarDevision, which main product was the office suite „StarOffice“. In July 2000 Sun announced an open source project , which was called „OpenOffice.org“. On October 13th , 2001 the „OpenOffice.org“

homepage went online and offered the chance to download the source code of „StarOffice 5.2 “. In October 2001 the first running version, called „Build 638c“ was released. The second version, named „OpenOffice.org 1.1“ was released in September 2003. The latest version „OpenOffice 2.0.2“ was published on 8th March 2006. The next version, which is called „OpenOffice 2.0.3“ is announced for June 2006. [Wiki06]

3.2 OpenOffice.org 2 Components

Writer

The Writer is a tool to create professional documents, memos, newsletters, WebPages and booklets. It offers a great variety of formatting text, inserting graphics, tables, diagrams, and different styles. [Open06]

Calc

With this application it's possible to create spreadsheets to calculate, analyse and present data fast and efficient (e.g. with diagrams). [Open06]

Draw

You can use sdraw to make drawings and shapes in different ways. [Open06]

Impress

Impress is based on draw, but you can also make nice presentations. [Open06]

Base

Beside the classical components OpenOffice.org has a database module where all kinds of databases can be created. It's also possible to set up a connection to other databases via ODBC11 or JDBC12. [Open06]

Math

Math is a tool for mathematical equations. It is most commonly used as an equation editor for text documents, but it can also be used with other types of documents or stand-alone. [Open06]

3.3 Architecture

The office suite OpenOffice.org is designed as a client server application, which communicates via TCP/IP. Typically both client and server components are installed on one computer, and the client uses the local server component. Due to this architecture it is in principle possible that a local client uses a remote server on a different machine. The OOo consists of different UNO components, which offer different functionalities.

By combining some of these components you will become a whole application like the Calc application. This also means that components can be re-used by different applications. [Flat05-1, p.4]

In Figure 1 the usage of the UNO component principle is shown.

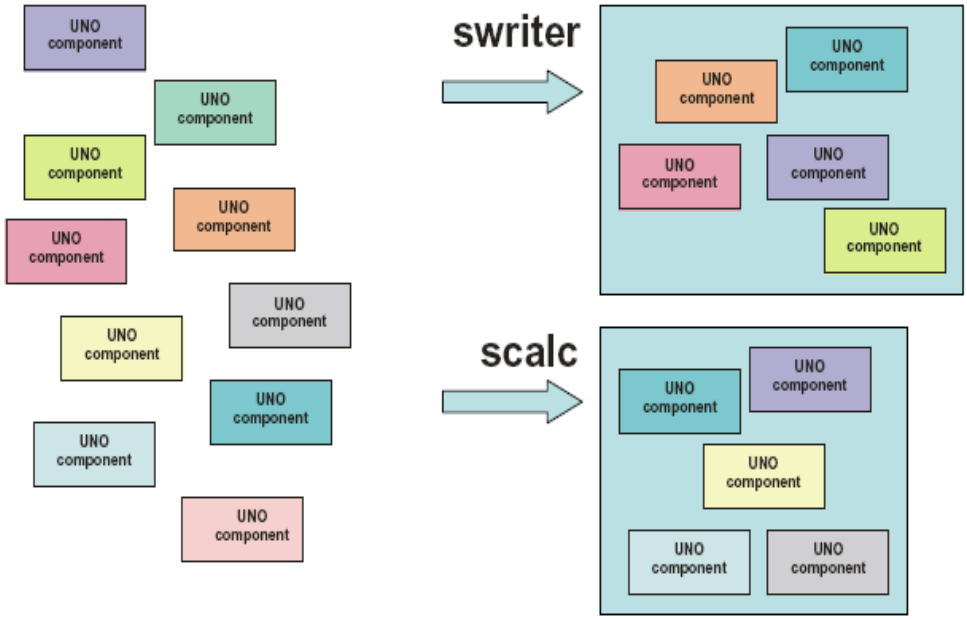


Figure 1: Configuring OpenOffice.org Applications from UNO components [Flat05-1, p.5].

3.3.1 Universal Network Object

Because each component is described in the interface description language (IDL) module, it is possible to implement UNO components in different programming languages. The UNO Interface Description Language Module can be described as:

> *„IDL modules may contain nested IDL modules, where the structure represents a hierarchy having a root module. Identifying a type in this hierarchy of modules is therefore easy, one starts out at the root module and names all nested modules one needs to traverse, leading in and separating the names with double colons (::, c-style) or separating them with a dot only (Java style). Hence the type named "XPrintable" has the fully qualified name "::com::sun::star::view::XPrintable" (C++) or "com.sun.star.view.XPrintable" (Java).“ [Flat05-1, p.4]*

There are three main advantages when using UNO components:

- different programming languages:
  As mentioned above different programming languages and scripting languages can be used to automate OpenOffice.org.

- different operating systems:
  OpenOffice.org is available and can be used on fifferent operating systems like Solaris, Linux or Windows.

- different networks:
  As mentioned in chapter 3.3 (see p.12) OOo is designed as client server application. Therefore it is possible to run OpenOffice over a network on different machines (see Figure 2).

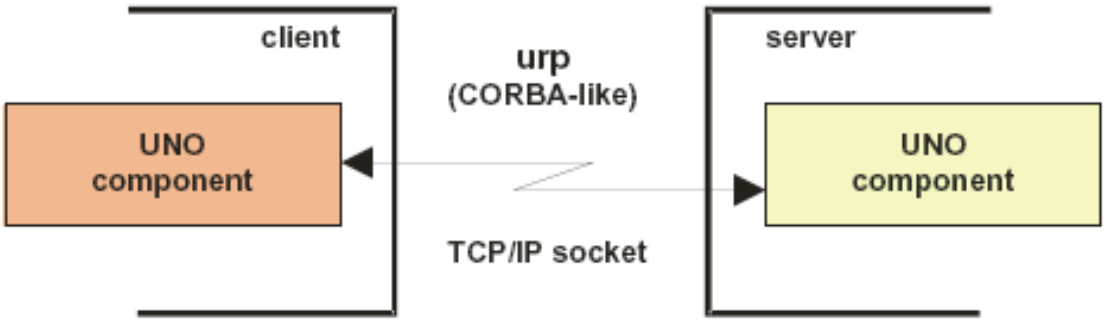


Figure 2: Client-/Server-Communications with UNO Components Using TCP/IP Sockets and the CORBA-like Protocol "urp" (UNO Remote Protocol). [Flat05-1, p.7]

3.3.2 UNO Service Components

> *„Each UNO component (defined as an IDL type in a module) usually represents a specific "service", consisting of additional services, and possessing properties and interfaces (usual offering methods and properties to a specific aspect of the UNO component) to it. Properties allow the storing of information appertunant to services." [Flat05-1, p.5]*

To obtain an instance of a service component a so-called „Service Manager" is needed.

3.3.2.1 Service Manager

The Service Manager is also called „factory" because it can be seen as a root component that creates and provides instances of service components, which can have a service manager themselves. Each Service Manager consists in a specific component context, e.g. the calc component. This concept is shown in Figure 3.
A often used service is the *com.sun.star.Desktop* service, which is used to load documents, to get the current document, and to access loaded documents. A service is instantiated by using its methods "createInstance()" or "createInstanceWithArguments()" and it is initiated by its fully qualified name. The returned object is called a „service object". [Burg06, p.17f]

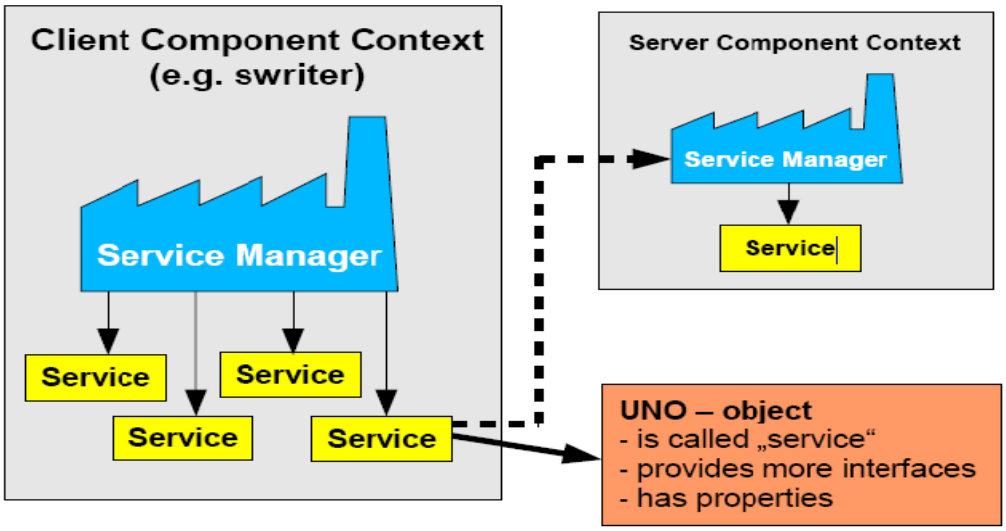


Figure 3: Service Manager [Burg06, p.17]

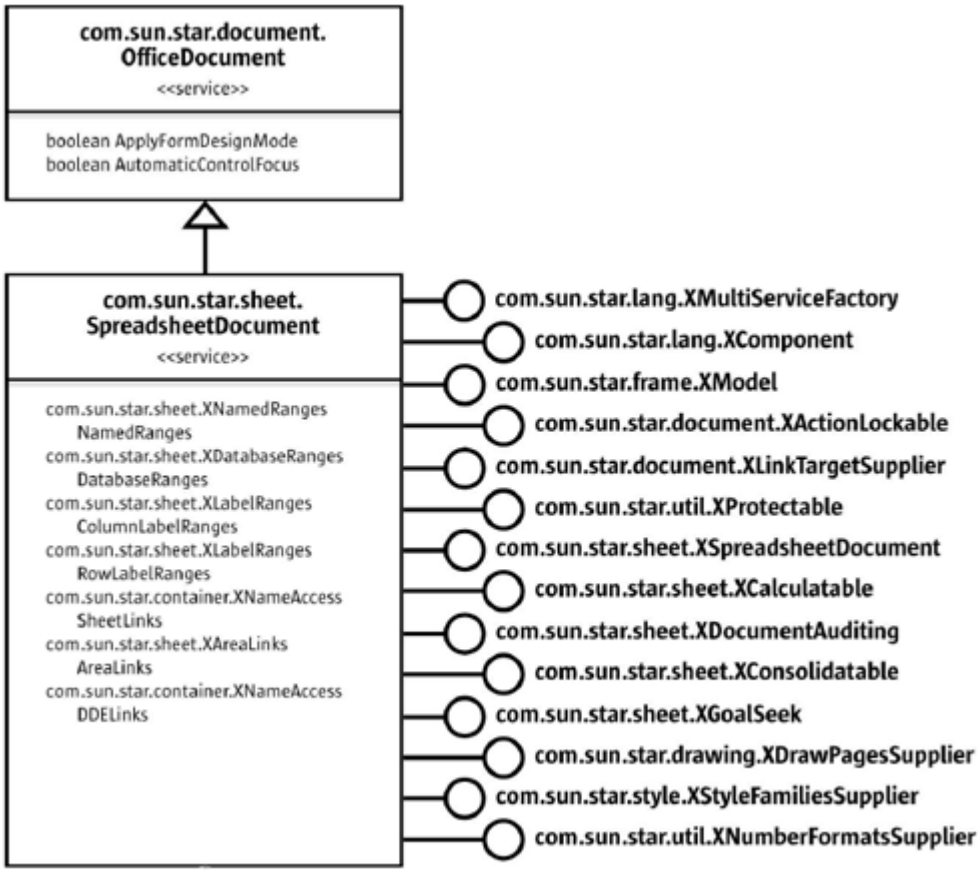


Figure 4: The SpreadsheetDocument service[Deve05, p.594]

In Figure 4 the SpreadsheetDocument service is shown in UML notation. It includes the OfficeDocument service and provides different interfaces like the interfaces XSpreadsheetdocument, XProtectable or XDrawPageSupplier. Note that each interface name begins with a X.

3.3.2.2 Interfaces

> *„An interface specifies a set of attributes and methods that together define one single aspect of an object.“ [Deve05, p.39]*

Interfaces like the XSpreadsheetDocument can be described as a collection of methods and optionally arguments. The XSpreadsheetDocument interface provides for example the method “getSheets()“ and returns the collection of sheets in the document. [Deve05, p.594]

3.3.3 UNO Java Access

After Sun took over StarDevision Java adapters were implemented to allow Java to interact with UNO components as if they were native Java components. Furthermore this infrastructure allows creating and implementing UNO components fully in Java. [Flat05-1, p.9]

44 Bean Scripting Framework (BSF)

The Bean Scripting Framework can be seen as glue between script languages and Java. With BSF a script language like JavaScript or ooRexx gets the ability to access Java objects and their methods. In addition it enables Java to execute programmes that are written in a supported scripting language. [Jaka06]

BSF supports several scripting languages: [Jaka06]

- JavaScript (using Rhino ECMAScript, from the Mozilla project)
- Python (using either Jython or JPython)

- Tcl (using Jacl)
- NetRexx (an extension of the IBM REXX scripting language in Java)
- XSLT Stylesheets (as a component of Apache XML project's Xalan and Xerces)

In addition, the following languages are supported with their own BSF engines: [Jaka06]

- Java (using BeanShell, from the BeanShell project)
- JRuby
- JudoScript
- Groovy
- ObjectScript

In addition there was another BSF created to support the scripting language ObjectRexx:

- BSF4Rexx

The Bean Scripting Framework consists of two main components, which are shown in Figure 5.

- The BSFManager

  The BSFManager is responsible for all scripting execution engines. To allow a Java application access to scripting services, an instance of the BSFManager has to be created first. Furthermore the BSFManager maintains the Java object registry and allows scripting languages to access Java. [Jaka06-1]

- The BSFEngine

  „*The BSFEngine provides an interface that must be implemented for a language to be used by BSF. This interface provides an abstraction of the scripting language's capabilities that permits generic handling of script execution and object registration within the execution context of the scripting language engine.*“ [Jaka06-1]

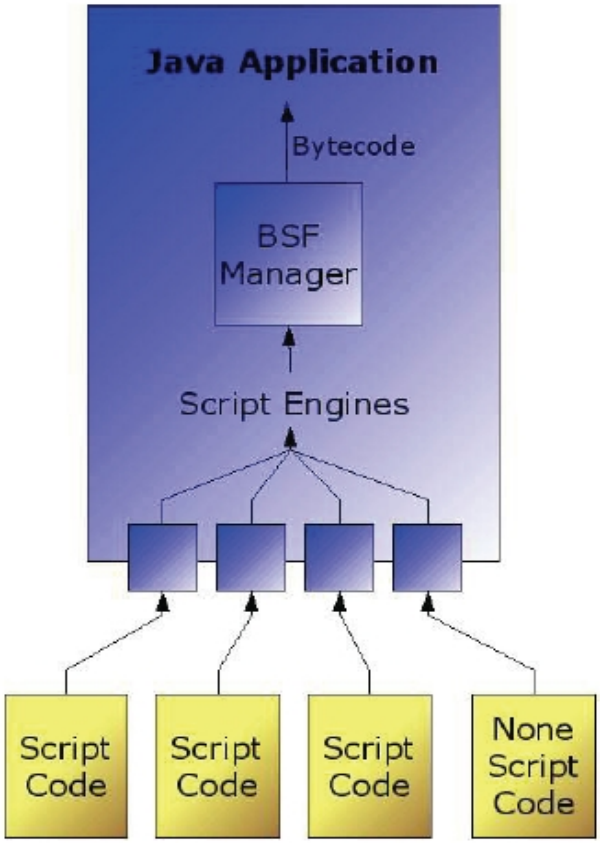


Figure 5: BSF Architecture [Hane05, p.17]

4.1 Bsf4Rexx

4.1.1 History

The first version of BSF4Rexx, which was called „Essener Version“, was developed in 2000 at the university of Essen by Prof. Mag. Dr. Rony G. Flatscher based on a proof of concept by his student Peter Kalender. In spring the first complete version of BSF4Rexx was presented to the RexxLa.

In 2002/2003 the second version called „Augsburger Version“ was developed. The main features of this version were the ability to load Java on Windows and Linux platforms. Additionally some bugs were fixed and some external Rexx functions were added.

The latest version is called „Vienna Version“ (see Figure 6) and was developed in 2003 but there is an ongoing development in progress. This version does not require strict Java types anymore and a lot of new functions for automating OOo were implemented. [Flat06-3]

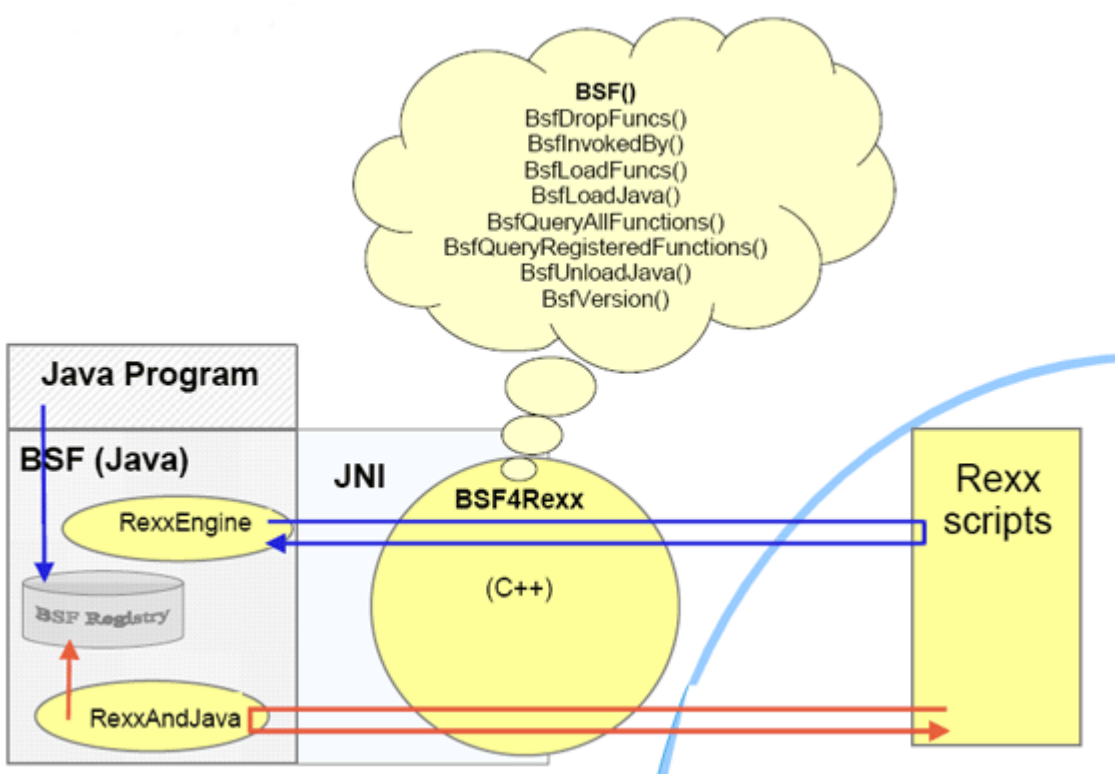


Figure 6: BSF4Rexx Architektur – Wiener Version [Flat06-3]

There are two main modules within the BSF4Rexx:

- BSF.CLS
- UNO.CLS

4.1.2 BSF.CLS

The module BSF.CLS is able to camouflage Java as Object Rexx and therefore enables ooRexx to import and use Java classes as if they were ooRexx classes. So Java can be seen as a huge ooRexx class library.

The module also provides public routines, classes and the environment symbol

“.BSF4REXX“. [Flat05-1, p.12]

4.1.3 UNO.CLS

The module UNO.CLS, which was formerly named OOo.CLS, generically supports the UNO component model. The module makes it easier to communicate with Open Office.org because it can automate common steps, like retrieving a DesktopObject. Therefore it uses the BSF.CLS. [BSF406]

Here are some routines of the UNO.CLS module which are used in the following examples:

- uno.createDesktop([context])
... creates and returns the reference to the OOo desktop object of the supplied"context" (could be any OOo server); if "context" is omitted, then the local OOo is used (ie. OOo installed on the machine the program runs on; no need to define ports explicitly)

- uno.connect(url_string)
... allows connecting to a remote OOo by using the URL according to the OOo documentation; returns a reference to the remote object that you denoted in the "url_string"

- convertToUrl(string)
... converts the fully qualified path to its URL representation as expected by OOo (takes differences between Linux and Windows transparently into account)

- uno.loadClass(java_class_name[, short_name])
... imports the UNO Java class "java_class_name" (as an UNO_PROXY) and stores it under the name "short_name" in the directory ".uno"; if "short_name is omitted then the unqualified class name (after the last dot) is used instead

- uno.setCell(xSheet, x, y, value)

... simple utility routine: inserts "value" (number, text or formula) at the 0-based co-ordinates "x" and "y" in the spreadsheet-object "xSheet" [BSF406]

55 Overall Concept

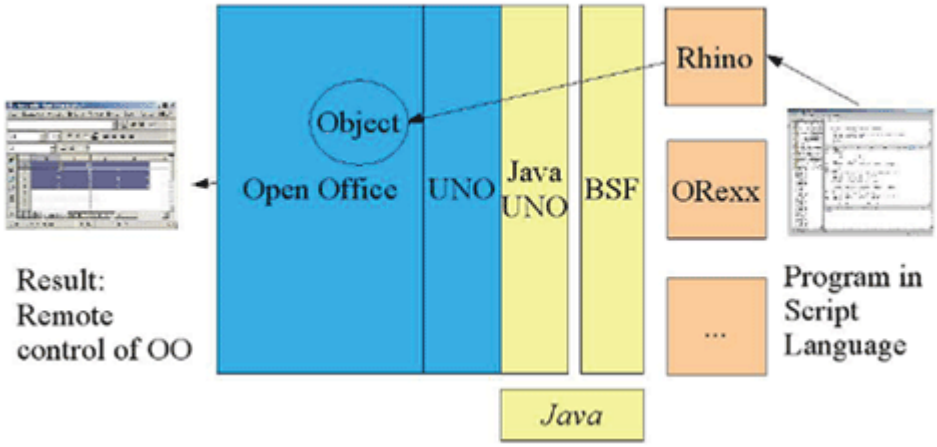


Figure 7: Overall Concept [Hahn05]

To automate OOo a scripting language is needed that is compatible to the Bean Scripting Framework (BSF) or provides its own BSF, like ORexx using BSF4Rexx. In Figure 7 the scripting language Rhino is used to describe the model.

The next step that has to be done is to convert a scripting language code to a Java-based one. This is realised with the BSF4Rexx module.

Because OOo is based on UNO (Universial Network Object) components, it is necessary to to gain access to these components. By using the Java Adapter it is possible to build a bridge between OOo and the programming language Java.

You can find the latest BSF4Rexx distribution at:
http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/

66 Installation Guide

This chapter will describe were you can retrieve and how to install the single components.

6.1 Install OpenOffice.org

You can retrieve the latest stable version of OpenOffice.org, at the time of writing the latest version is 2.0.2, by downloading it from the OpenOffice.org homepage: http://www.openoffice.org/
There you can choose a operating system and a language you like.

6.2 Install Open Object Rexx

Open Object Rexx can be downloaded from the ooRexx hompage at http://www.oorexx.com/download.html.
At the time time of writing there are two versions available. Version 3.0 is the current stable version but it isn't available for AIX. Version 3.1 is the current beta release but isn't available for Solaris at the moment. [Oore06]

6.3 Install Java

To check if Java is installed on your machine, open a command window and type in:
java -version

This will print the actually version of Java to the command window, e.g.:

j*ava version "1.5.0_06"*

Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)

If Java isn't installed or if your version is smaller than 1.4, get a new one by downloading it from the sun homepage: http://java.sun.com

You can choose between the this two versions:

- Java runtime version (JRE - Java runtime environment)
- Java developer version (JDK - Java development kit)

[BSF406-1]

6.4 Install BSF4Rexx

First you have to download the archive „BSF4Rexx_install.zip" which can be obtained at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/ and unzip it.
After downloading and unzipping the package you have to run the script „rexx setupBSF.rex". This will create some new files including the file „installBSF4Rexx.cmd" which has to be executed. Now you should be able to run BSF4Rexx scripts using the dispatcher program „rexxj.cmd" on Windows or „rexxj.sh" on Linux.
If any problem occurred you can take a look at the „readmeBSF4Rexx.txt" file, which is included in the „BSF4Rexx" package and provides an detailed installation guide. [BSF406-1]

6.5 ConFigure OpenOffice.org

The next step that has to be performed is to enable Java in OpenOffice.org.
Therefore start an OOo application like the calc component and choose the menu „Tools" the option „Options". Then choose „OpenOffice.org >> Java". Now enable a Java Runtime environment by clicking the checkbox "Use a Java runtime-environment" and choose a appropriate version (see Figure 8). [BSF406-1]

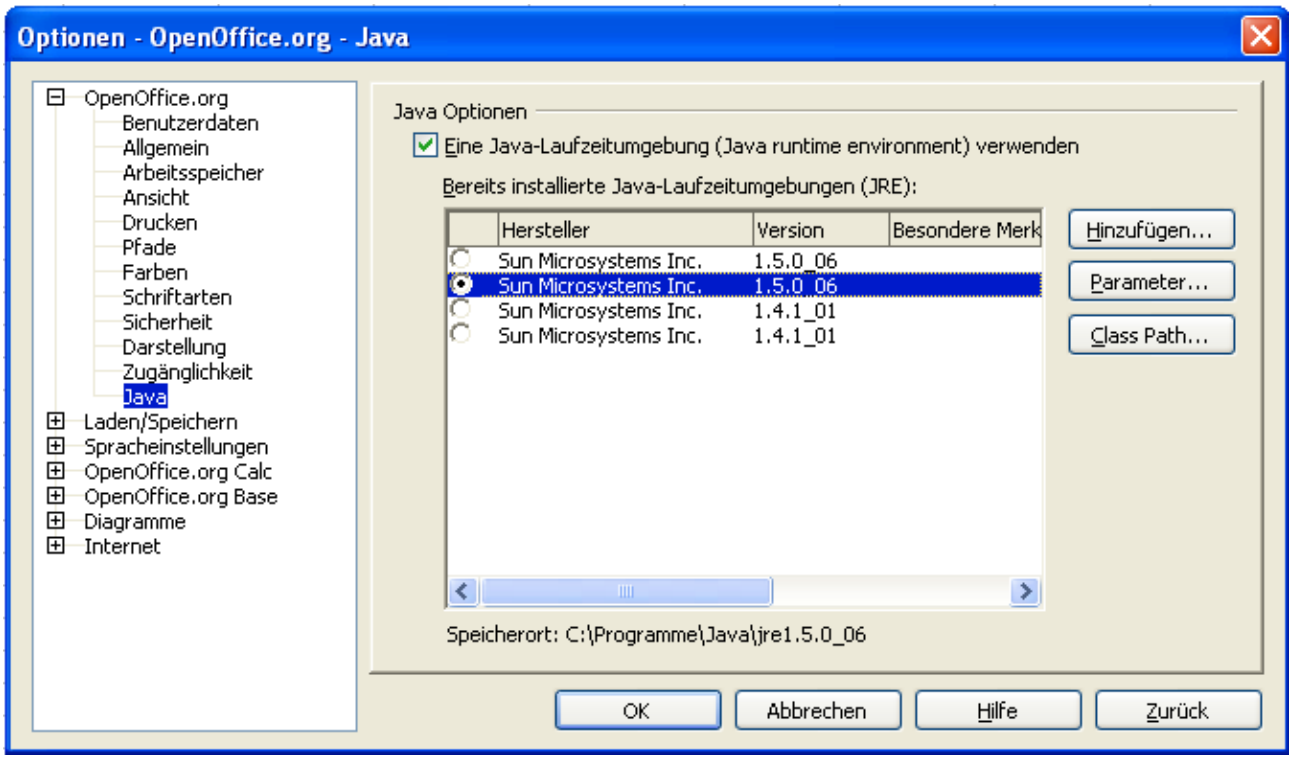


Figure 8: Enable Java in OOo

The next step is to run "rexx setupOOo.rex path-to-OOo-directory" like "rexx setupOOo.rex d:\Programme\OpenOffice.org 2.0".

This will create a script named "setEnvironment4OOo.cmd" ("setEnvironment4OOo.sh" on Linux)that contains the definitions for the environment variable "CLASSPATH". Whenever you want to run a Rexx script from the command window, which automates OOo, run "setEnvironment4OOo.cmd" for Windows or "../setEnvironment4OOo.sh" for Linux first and then start the Rexx script. It is also possible to set the environment variable permanently. [BSF406-1]

77 The Calc Component

Figure 9 shows the Spreadsheet Document Model, which is needed when working with the calc component. The Spreadsheet Document Model consists of five major architectural areas.

- Spreadsheets Container

  When working with the calc component almost everything happens in a spreadsheet. A spreadsheet is contained in the spreadsheet container and can be extracted from it.

- Service Manager (document internal)

  The service manager of the spreadsheet document model is responsible for creating shape objects, text fields for page headers and form controls which can be added to a spreadsheet. The document service manager is different from the service manager which was described in chapter 3.3.2.1 (see p.14) and which is needed to connect to the office. There exists a service manager for each document model (e.g. the writer document model or the spreadsheet document model)

- DrawPages

  A draw page can be descried as a transparent layer that lies upon a sheet and contains drawing elements.

- Content Properties

  Content properties allow access to the linked and named contents of all sheets. here are no content suppliers as in text documents, because the actual content of a spreadsheet document lies in its sheet objects.

- Objects for Styling

Objects for Styling are services that are responsible for document wide styling and structuring of the spreadsheet document. For example there are style family suppliers for cells and pages.

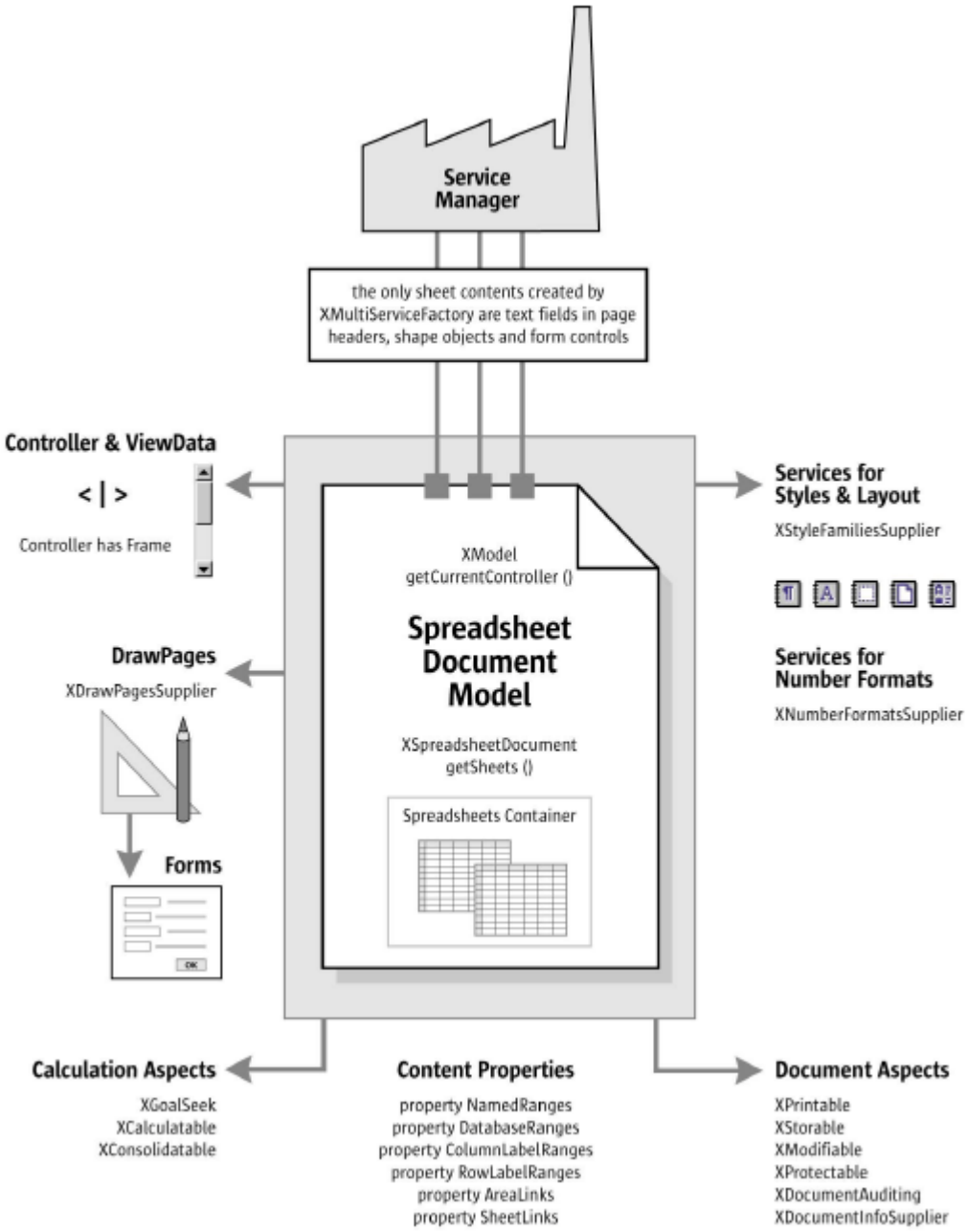


Figure 9: Spreadsheet Document Component[Deve05, p.584]

7.1 Main Services

Figure 10 illustrates the main elements of an spreadsheet. These elements are represented through services and they are mainly used in the following examples.

- com.sun.star.sheet.Spreadsheet
  represents a whole sheet

- com.sun.star.sheet.SheetCellRange
  represents a range of cells whitin a sheet

- com.sun.star.sheet.SheetCell
  represents a single cell

- com.sun.star.table.TableColumn
  represents the colums of a sheet

- com.sun.star.table.TableRow
  represents the rows of a sheet

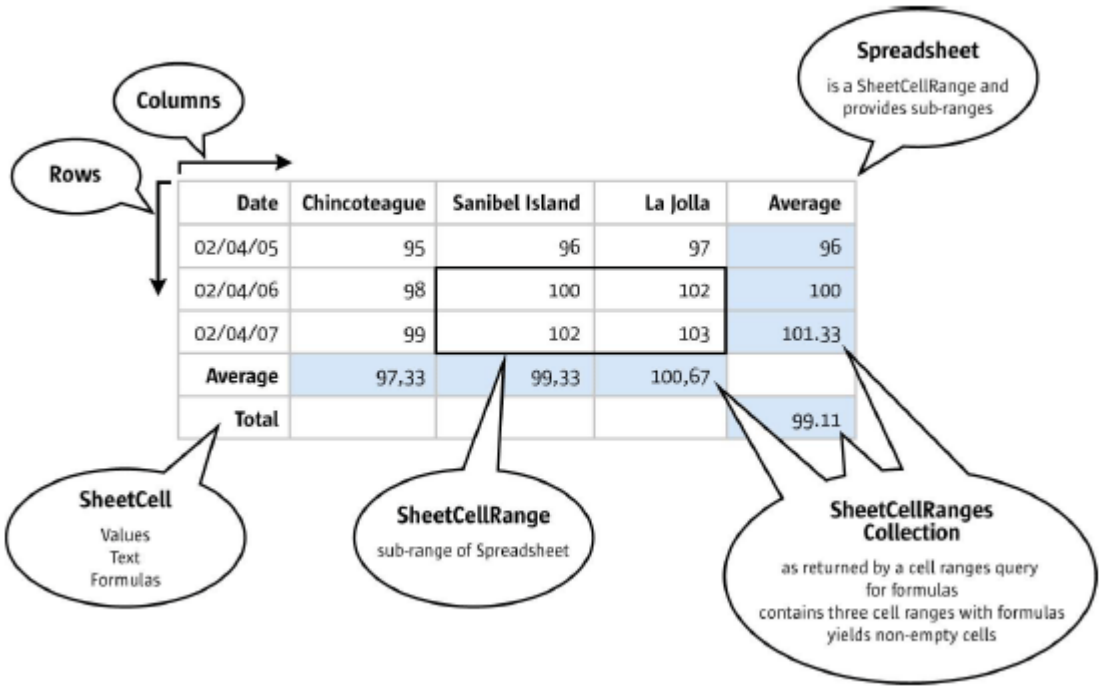


Figure 10: Main Spreadsheet Services[Deve05, p.597]

7.2 Examples

The examples below demonstrate the automation of OpenOffice.org and its calc component. They should demonstrate which interfaces are availiable in the calc component context and how they are used. The source code of each example will be described in more detail to show which steps have to be performed. The code can easily be reused by using copy and paste. Furthermore these examples should help to create a database for OOo automation using ooRexx.

The most examples are taken from the OOo code snippet base (http://codesnippets.services.openoffice.org/) [Code06], where you can find snippets written in OOBasic, Java, C++ and Phyton, and have been translated into ooRexx.

Additional examples, which were originally written in Java, are taken from the OOo Developers Guide [Deve05], which can be downloaded at: http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html

If someone wants to know more about specific services or interfaces or wants to create own programmes then have a look into the OOo Api which can be found at: http://api.openoffice.org/ [ApiO06]

7.2.1 Example01 - HelloWorld

This example creates a new Calc document, retrieves the first sheet and inserts a string into the first cell.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet, 0, 0, "HelloWorld!"



::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 11.

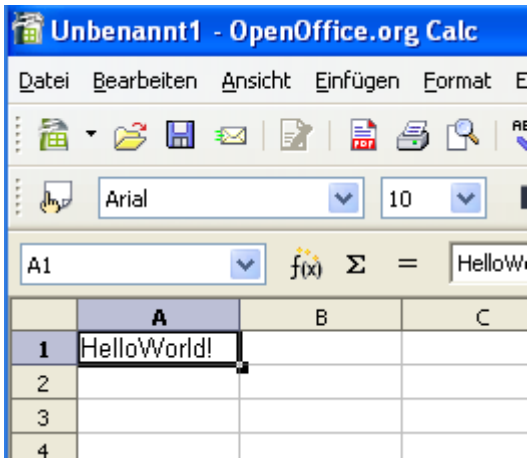


Figure 11: HelloWorld

The lines of code explained in more detail:

In Cutout.1 a connection is set and a XDesktop object is retrieved.

```
Cutout.1
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface
```

Now (Cutout.2) the method loadComponentFromUrl, with its values (URL, TargetFrame, SearchFlag, PropertyValue), which is provided by the XComponentLoader interface is needed to open a existing or new file.

The URL is an important attribute because it specifies whether a new file or a existing file will be opend. Furthermore it is used in each example.

URL

url = "private:factory/scalc"	-- opens a new calc document
url = "private:factory/swriter"	-- opens a new writer document
url = "private:factory/simpress"	-- opens a new impress document
url = "private:factory/sdraw“	-- opens a new draw document
url = "private:factory/sdatabase"	-- opens a new database
url = "private:factory/smath"	-- opens a new math document
url = „http://www.wu-wien.ac.at“	-- opens an html document
url = „file:///c:/test.odt“	-- opens an existing document

```
Cutout.2
/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)
```

In the next step (Cutout.3) the interface XspreadsheetDocument and its method getSheets() is invoked to get the collection of sheets in the document. In this example getByIndex() from the *com.sun.star.container.XIndexAccess* interface is used to retrieve a sheet.

It is also possible to get a sheet by its name. This is done in example 03, Cutout.1 (p.32).

```
Cutout.3
/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet, 0, 0, "HelloWorld!"
```

In cutout 3 the public routine „UNO.setCell“ is invoked. It sets the value of a specific cell (0,0 -- „A1“) to the specified value (HelloWorld!).

7.2.2 Example02 - Merging Cells

This example merges cells.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet, 0, 0, "Student"

CALL syssleep 3
/* create and retrieve a CellRange*/
xCellRange = xSheet~xCellRange~getCellRangeByName("A1:A5")

/* merge the cells */
xMergeRange = xCellRange~xMergeable
xMergeRange~merge(.true)

::requires UNO.cls      -- get UNO support
```

The result can be seen in Figure 12.

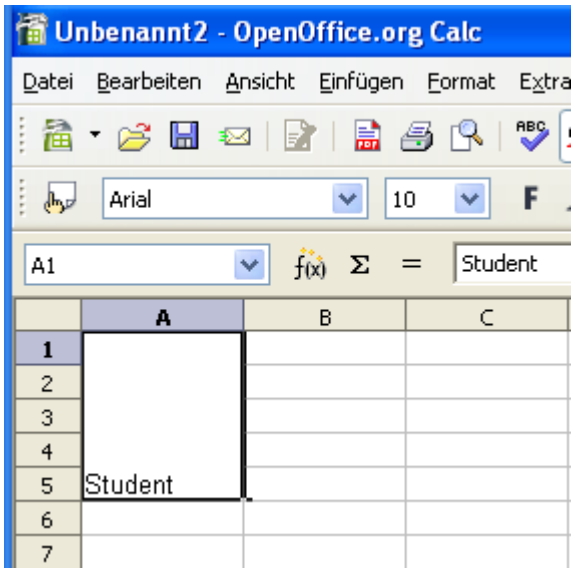


Figure 12: Merging Cells

The lines of code explained in more detail:

After creating and retrieving a new spreadsheet document the predefined function „syssleep“(Cutout.1) is called, which will stop the system for 3 seconds to make the change visible.

```
Cutout.1
CALL syssleep 3
```

In cutout 2 a new cell range is retrieved using the XcellRange interface.

```
Cutout.2
/* create and retrieve a CellRange*/
xCellRange = xSheet~xCellRange~getCellRangeByName("A1:A5")
```

To merge the cells of the *XCellRange* the interface *XMergeable* is needed (Cutout.3).

```
Cutout.3
/* merge the cells */
xMergeRange = xCellRange~xMergeable
xMergeRange~merge(.true)
```

7.2.3 Example03 - Copy a Sheet

This example copies the active sheet and focuses the new one.

```
/* get the dsktop (an Xdesctop object) */
oDesktop        = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      -- get componentLoader interface


/* open the file: test.ods  */
url = ConvertToURL(directory()"/test.ods")
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

/* get  sheet „Rexx“ in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XNameAccess~getByName("Rexx")~XSpreadSheet

/* insert  values into cells */
CALL UNO.setCell xSheet, 0, 0, "This sheet will be copied"
CALL UNO.setCell xSheet, 0, 1, "333"
CALL UNO.setCell xSheet, 0, 2, "222"
CALL UNO.setCell xSheet, 0, 3, "111"

CALL syssleep 3

/* copy sheet */
xSheets = xDocument~getSheets()
xSheets~copyByName("Rexx", "rexx2", 2)

/* set focus on new sheet */
xFocusSheet = xDocument~getSheets~XNameAccess~getByName("rexx2")~XSpreadSheet
xController = xDocument~XModel~getCurrentController
xSpreadsheetView = xController~xSpreadsheetView~setActiveSheet(xFocusSheet)


::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 13.

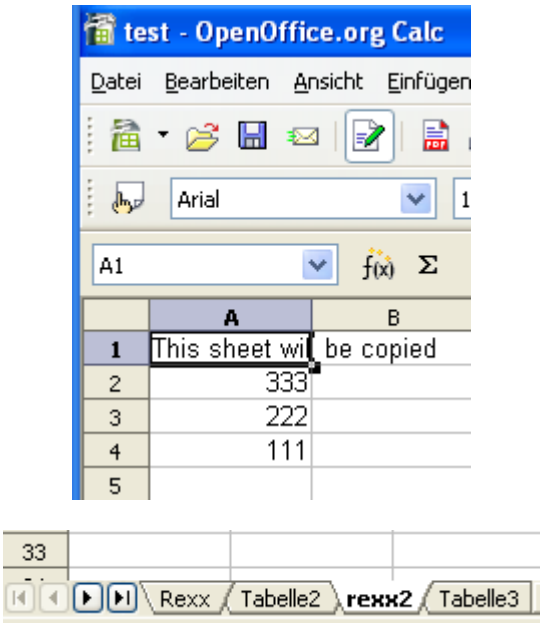


Figure 13: Copy a Sheet

The lines of code explained in more detail:

In Cutout.1 the interface *com.sun.star.container.XNameAccess* is used to get a sheet by its name.

```
Cutout.1
/* get  sheet „Rexx“ in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XNameAccess~getByName("Rexx")~XSpreadSheet
```

The *copyByName()* method (Cutout.2) is responsible for copying a sheet and for inserting a new sheet. The first value the method needs is the name of a existing sheet that you want to copy. The second value creates a sheet with its given name and the third value specifies the position the new sheet is inserted into.

```
Cutout.2
xSheets~copyByName("Rexx", "rexx2", 2)
```

The controller (Cutout.3), which is invoked by the method *getCurrentController()* from the *XModel* interface, provides the interface *XSpreadsheetView*. With its method

setActiveSheet() it is possible to focus a specified sheet.

```
Cutout.3
xController = xDocument~XModel~getCurrentController
xSpreadsheetView = xController~xSpreadsheetView~setActiveSheet(xFocusSheet)
```

7.2.4 Example04 - Set Cell Attributes

This Example changes the char weight and the cell background colour.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* get cell address */
xCell = xSheet~getCellByPosition(0, 0)
xCell2 = xSheet~getCellByPosition(0, 1)

/* set cell properties */
xCell~xPropertySet~setPropertyValue("CharWeight", box("float",  -
bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))

xCell2~xPropertySet~setPropertyValue("CellBackColor", box("int", "CCCCCC"x ~c2d))

/* insert  values into cells */
CALL UNO.setCell xSheet, 0, 0, "CharWeight"
CALL UNO.setCell xSheet, 0, 1, "CellBackColor"

::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 14.

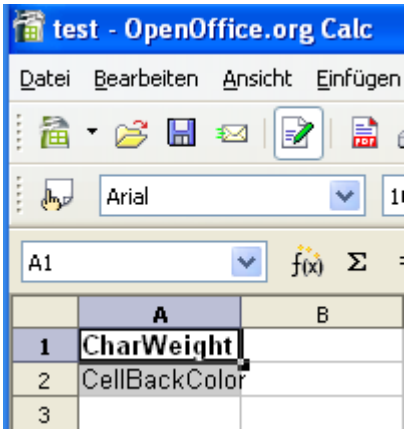


Figure 14: Set Cell Attributes

The lines of code explained in more detail:

In Cutout.1 it is shown how to obtain a cell address by its position. The parameters 0,0 points to the first cell in the first row („A1“). The cell address is needed to manipulate its appearance (Cutout.2).

```
Cutout.1
/* get cell address */
xCell = xSheet~getCellByPosition(0, 0)
xCell2 = xSheet~getCellByPosition(0, 1)
```

Now (Cutout.2) it is possible, through the *XPropertySet* interface, to modify the properties of a cell.

```
Cutout.2
/* set cell properties */
xCell~xPropertySet~setPropertyValue("CharWeight", box("float", -
bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))

xCell2~xPropertySet~setPropertyValue("CellBackColor", box("int", "CCCCCC"x ~c2d))
```

7.2.5 Example05 - Set Column/Row Attributes

This example colours the 4th column and deletes the 5th row.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet, 0, 4, "This row will be deleted!"

/* get rows and columns */
xColumnRowRange = xSheet~xColumnRowRange~getColumns()
xColumn = xColumnRowRange~getByIndex(3)

xColumnRowRange = xSheet~xColumnRowRange~getRows()
xRow = xColumnRowRange~getByIndex(4)

CALL syssleep 3

/* set properties of columns and rows */
xColumn~xPropertySet~setPropertyValue("CellBackColor", box("int", "006666"x ~c2d))

CALL syssleep 4

xRow~xPropertySet~setPropertyValue("IsVisible", Boolean (false))

::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 15:

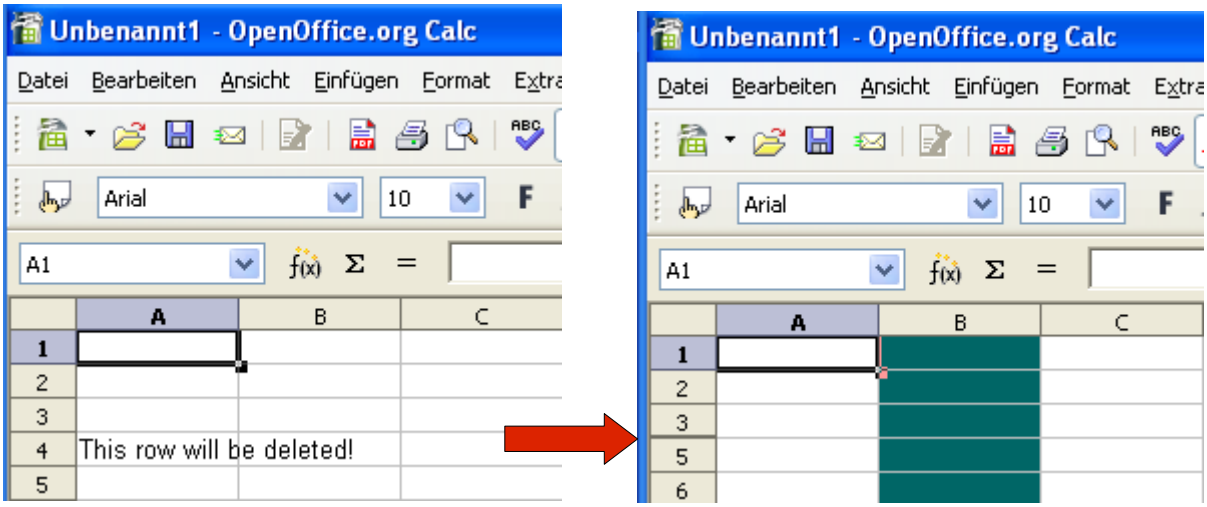


Figure 15 Set Column/Row Attributes

The lines of code explained in more detail:

The *XColumnRowRange* (Cutout.1) interface provides the access to the rows and columns of a sheet. If you want to get the collection of the columns you have to use the method *getColumns()*. The method *getRows()* returns the collection of rows. With the *getByIndex()* method a specified row or column can be accessed.

Cutout.1

```
/* get rows and columns */
xColumnRowRange = xSheet~xColumnRowRange~getColumns()
xColumn = xColumnRowRange~getByIndex(3)
```

The column and row properties (Cutout.2) can be modified the same way like the cell properties in example05, Cutout.2 (p.40).

Cutout.2

```
/* set properties of columns and rows */
xColumn~xPropertySet~setPropertyValue("CellBackColor", box("int", "006666"x ~c2d))
```

7.2.6 Example06 - Insert an image

This following example uses the draw page to inserts an image into the active sheet.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* get DrawPage */
xDrawPages = xSheet~xDrawPageSupplier
xDrawPage = xDrawPages~getDrawPage~xDrawPage

/*create shape to insert picture*/
xFactoryManager = xCalcComponent~XMultiServiceFactory
calcShape = xFactoryManager~createInstance("com.sun.star.drawing.GraphicObjectShape")
xcalcImg = calcShape~xShape

size = .bsf~new("com.sun.star.awt.Size")              -- set size
point = .bsf~new("com.sun.star.awt.Point")
size~Height = 2500
size~Width = 8000
point~x = 1000
point~y= 1000

xcalcImg~setSize(size)
xcalcImg~setPosition(point)

url1 = ConvertToURL(directory()"/oorexx.jpg")
xcalcImg~xPropertySet~setPropertyValue("GraphicURL", url1)

xDrawPage~add(xcalcImg)        -- add image to page
::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 16.

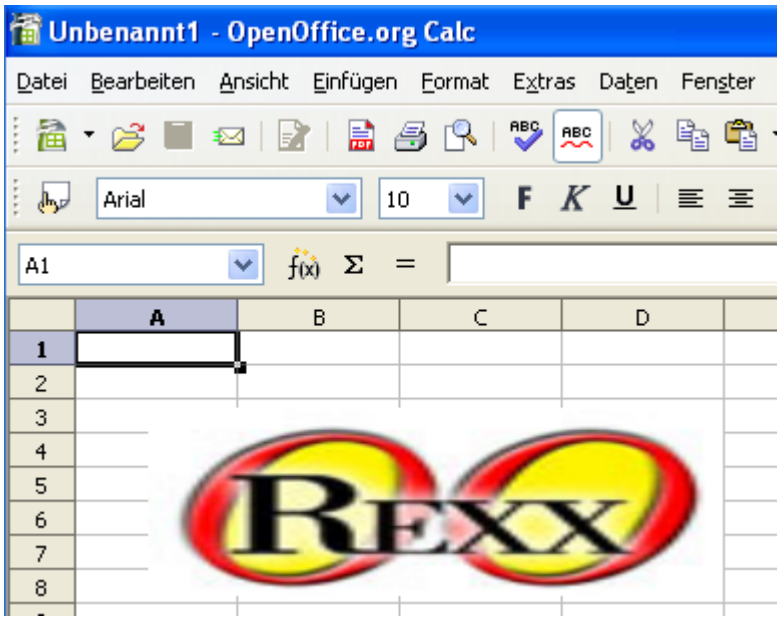


Figure 16: Insert an Image

The lines of code explained in more detail:

In Cutout.2 the *XDrawPageSupplier* interface, which is needed to insert new shapes, is invoked.

```
Cutout.1
/* get DrawPage */
xDrawPages = xSheet~xDrawPageSupplier
xDrawPage = xDrawPages~getDrawPage~xDrawPage
```

Now the Service Manager of the current document (Cutout.2) is necessary to create an instance of the *com.sun.star.drawing.GraphicObjectShape.*

```
Cutout.2
/*create shape to insert picture*/
xFactoryManager = xCalcComponent~XMultiServiceFactory
calcShape = xFactoryManager~createInstance("com.sun.star.drawing.GraphicObjectShape")
xcalcImg = calcShape~xShape
```

In Cutout.3 the URL to the image resource is set and the image is added to the sheet.

```
Cutout.3
url1 = ConvertToURL(directory()"/oorexx.jpg")
xcalcImg~xPropertySet~setPropertyValue("GraphicURL", url1)

xDrawPage~add(xcalcImg)        -- add image to page
```

7.2.7 Example07 - Auto Format

This example adds an auto format to a cell range.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

CALL syssleep 3

/* insert  value into cells */
CALL UNO.setCell xSheet, 0, 0, "Something"
CALL UNO.setCell xSheet, 0, 2, "New"
CALL UNO.setCell xSheet, 0, 4, "Old"
CALL UNO.setCell xSheet, 2, 0, "12"
CALL UNO.setCell xSheet, 2, 4, "43"
CALL UNO.setCell xSheet, 2, 2, "6"
CALL UNO.setCell xSheet, 4, 0, "17"
CALL UNO.setCell xSheet, 4, 4, "49"
CALL UNO.setCell xSheet, 4, 2, "66"

CALL syssleep 3

/* get cell range */
xCellRange = xSheet~xCellRange~getCellRangeByPosition(0, 0, 5, 5)

/* apply auto format */
xAutoForm = xCellRange~XAutoFormatTable

xAutoForm~autoFormat("Gelb")

::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 17.

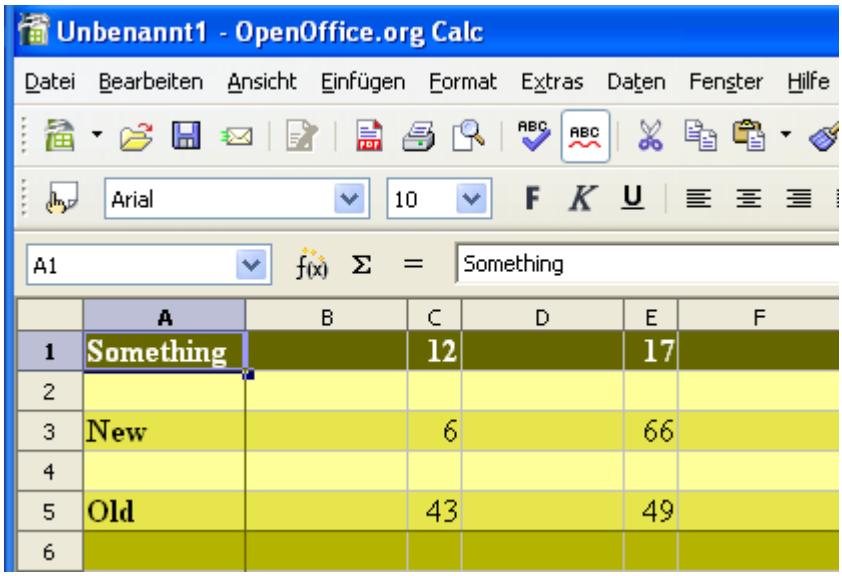


Figure 17: Auto Format

The lines of code explained in more detail:

First it is needed to get the cell range to which the format should be added. This is done by *getCellRangeByPosition(0, 0, 5, 5)*. The values are: *(first column, first row, last column, last row)*

```
Cutout.1
xCellRange = xSheet~xCellRange~getCellRangeByPosition(0, 0, 5, 5)
```

The interface *XAutoFormatTable* provides the Method *autoFilter()* which needs the name of the Filter you want to apply. Please note that this can differ from the language of your OOo installation.

```
Cutout.2
/* apply auto format */
xAutoForm = xCellRange~XAutoFormatTable
xAutoForm~autoFormat("Gelb")
```

7.2.8 Example08 - Filter

This example adds a filter to a cell range and hides all values that are less than 5.

```
oDesktop        = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader



url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

CALL UNO.setCell xSheet, 0, 0, "1"
CALL UNO.setCell xSheet, 0, 1, "2"
CALL UNO.setCell xSheet, 0, 2, "3"
CALL UNO.setCell xSheet, 0, 3, "4"
CALL UNO.setCell xSheet, 0, 4, "5"
CALL UNO.setCell xSheet, 0, 5, "6"
CALL UNO.setCell xSheet, 0, 6, "7"
CALL UNO.setCell xSheet, 0, 7, "8"

CALL syssleep 4

myRange = xSheet~XCellRange~getCellRangeByName("A1:A7")

xFilter = myRange~XSheetFilterable
xFilterDesc = xFilter~createFilterDescriptor(.true)

CALL UNO.loadClass "com.sun.star.sheet.TableFilterField"

/* creating an array wit filter criteria */
aFilterFields = bsf.createArray(.UNO~TableFilterField, 1)

aFilterFields[1] = .UNO~TableFilterField~new
aFilterFields[1]~Field = 0
aFilterFields[1]~IsNumeric = true
aFilterFields[1]~Operator = bsf.getConstant("com.sun.star.sheet.FilterOperator","GREATER_EQUAL")
aFilterFields[1]~NumericValue = 5
```

```
xFilterDesc~setFilterFields(aFilterFields)
xFilterDesc~xPropertySet~setPropertyValue("ContainsHeader", box("boolean", .false))
xFilter~filter(xFilterDesc)

::requires UNO.CLS
```

The result can be seen in Figure 18.

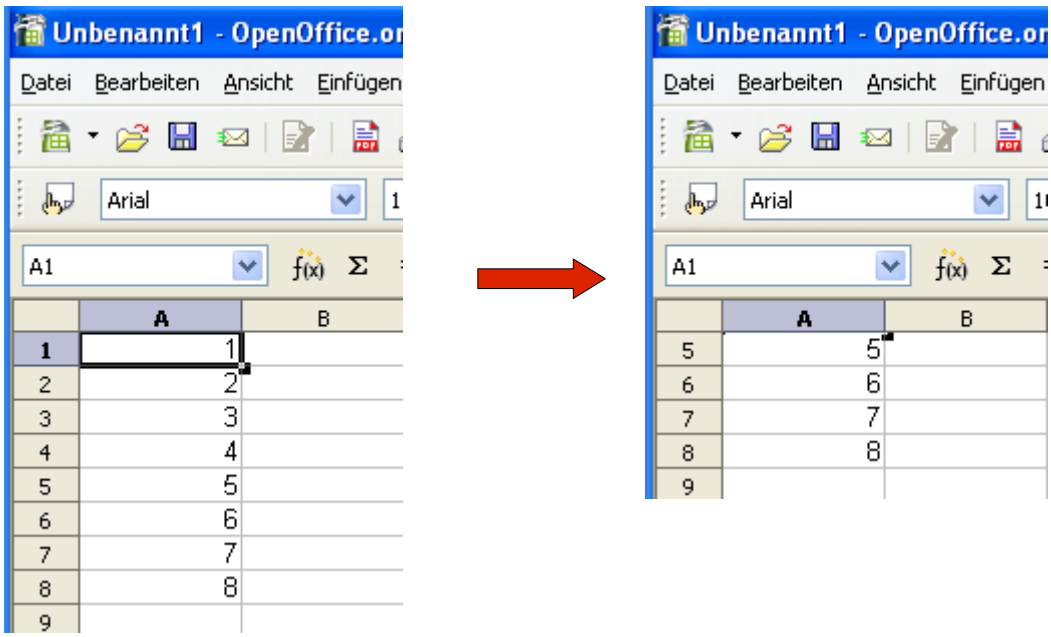


Figure 18: Filter

The lines of code explained in more detail:

The *com.sun.star.sheet.XSheetFilterDescriptor* (Cutout.1) interface is used to set the filter criteria as a sequence of *com.sun.star.sheet.TableFilterField* (Cutout.2) elements.

```
Cutout.1
xFilter = myRange~XSheetFilterable
xFilterDesc = xFilter~createFilterDescriptor(.true)
```

„com.sun.star.sheet.TableFilterField struct describes a single condition and contains the following members:

- *Connection has the values AND or OR, and specifies how the condition is connected to the pre vious condition in the sequence. For the first entry,*

Connection is ignored.

• Field is the number of the field that the condition is applied to.

• Operator is the type of the condition, such as EQUAL or GREATER

• IsNumeric selects a numeric or textual condition.

• NumericValue contains the value that is used in the condition if IsNumeric is true.

• StringValue contains the text that is used in the condition if IsNumeric is false.“ [Deve05, p.648]

The filter criteria (Cutout.2) are defined within an array.

Cutout.2

```
/* creating an array wit filter criteria */
aFilterFields = bsf.createArray(.UNO~TableFilterField, 1)

aFilterFields[1] = .UNO~TableFilterField~new
aFilterFields[1]~Field = 0
aFilterFields[1]~IsNumeric = true
aFilterFields[1]~Operator = bsf.getConstant("com.sun.star.sheet.FilterOperator","GREATER_EQUAL")
aFilterFields[1]~NumericValue = 5
```

7.2.9 Example09 - Header

This example will set the page header automatically to: „Rexx was here“

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet,  0, 0, "The header is set!"

/*create an instance of page style*/
xServiceManager = xDocument~XMultiServiceFactory
xPageStyle = xServiceManager~createInstance("com.sun.star.style.PageStyle")
xFamiliesSupplier = xDocument~XStyleFamiliesSupplier
xStyle = xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~XNameContainer

/*get the header and turn it on*/
xHeader = xStyle~getByName( "Default" )
xHeader~XPropertySet~setPropertyValue("HeaderIsOn", box("boolean", .true))

/*get the property value and set it*/
headerText = xHeader~XPropertySet~getPropertyValue("RightPageHeaderContent")
xHeaderText = headerText~XHeaderFooterContent

xHeaderText~getLeftText()~setString("Rexx")
xHeaderText~getCenterText()~setString("was")
xHeaderText~getRightText()~setString("here")
XHeader~XPropertySet~setPropertyValue("RightPageHeaderContent", xheaderText)

::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 19.

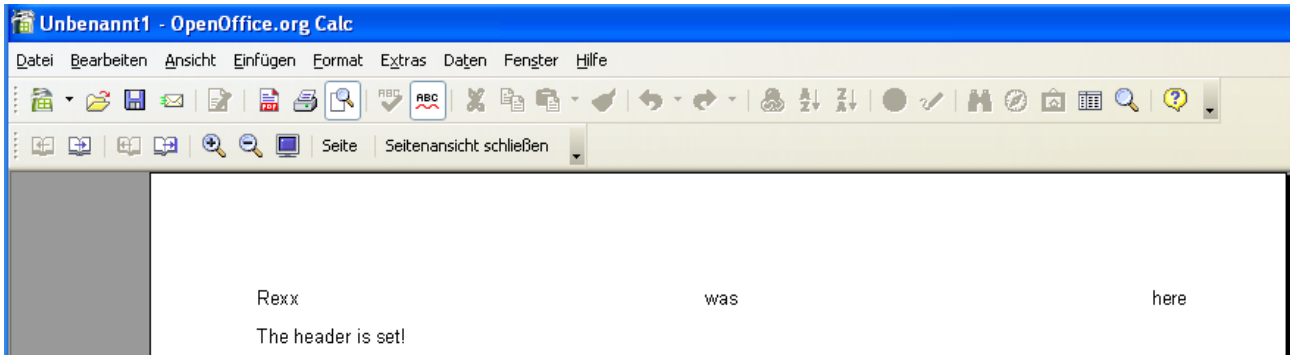


Figure 19: Header

The lines of code explained in more detail:

First the Service Manager of the current document (Cutout.1) is needed to retrieve the page style of a sheet which is responsible for the header.

```
Cutout.1
/*create an instance of page style*/
xServiceManager = xDocument~XMultiServiceFactory
xPageStyle = xServiceManager~createInstance("com.sun.star.style.PageStyle")
xFamiliesSupplier = xDocument~XStyleFamiliesSupplier
xStyle = xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~XNameContainer
```

Now it is possible to get the header which, is showed in Cutout.2.

```
Cutout.2
/*get the property value and set it*/
headerText = xHeader~XPropertySet~getPropertyValue("RightPageHeaderContent")
xHeaderText = headerText~XHeaderFooterContent
```

In Cutout.3 the method *getRightText()* and the method *setString()* are used to set the header's content.

```
Cutout.3
xHeaderText~getRightText()~setString("here")
XHeader~XPropertySet~setPropertyValue("RightPageHeaderContent", xheaderText)
```

7.2.10 Example10 - Page Size

This example changes the page size two times. First it switches the landscape and then sets the page to a manually given size. To see the result of this example you need to change the view by clicking this symbol from the menu bar: 

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

CALL UNO.setCell xSheet, 0, 0, "Please change the view"

xServiceManager = xDocument~XMultiServiceFactory

/*create an instance of page style*/
xPageStyle = xServiceManager~createInstance("com.sun.star.style.PageStyle")
xFamiliesSupplier = xDocument~XStyleFamiliesSupplier
xStyle = xFamiliesSupplier~getStyleFamilies~getByName("PageStyles")~XNameContainer
xPage = xStyle~getByName( "Default" )

xPageSize = xPage~XPropertySet~getPropertyValue("Size")

CALL syssleep 5

/*set size*/
height = xPageSize~Height
xPageSize~Height = xPageSize~Width
xPageSize~Width = height

/*apply size to page size*/
xPage~xPropertySet~setPropertyValue("Size",xPageSize)
xPage~xPropertySet~setPropertyValue("IsLandscape", box("new Boolean", (true)))

CALL syssleep 5
```

```
/*set new size*/
width = 5000
height = 10000

xPageSize~Height = height
xPageSize~Width=height = width

/*apply size to page size*/
xPage~xPropertySet~setPropertyValue("Size",xPageSize)
xPage~xPropertySet~setPropertyValue("IsLandscape", box("new Boolean", (true)))

::requires UNO.cls      -- get UNO support
```

The lines of code explained in more detail:

Here the page style is also needed like in example 09, Cutout.1 (p.50). But this time the property size is needed.

```
Cutout.1
xPageSize = xPage~XPropertySet~getPropertyValue("Size")
```

7.2.11 Example11 - Subtotal

This example adds a subtotal function to the active sheet and sums up the values of cell D2 and D3 and puts the total result in cell D4.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)


/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  value into cells */
CALL UNO.setCell xSheet, 3, 1, 100

myRange = xSheet~XCellRange~getCellRangeByName("A1:C6")

/*create SubTotalDescriptor( */
xSub = myRange~XSubTotalCalculatable
xSubDesc = xSub~createSubTotalDescriptor(.true)

CALL UNO.loadClass "com.sun.star.sheet.SubTotalColumn"

/*create array to set values*/
aColumn = bsf.createArray(.UNO~SubTotalColumn, 1)
aColumn[1] = .UNO~SubTotalColumn~new
aColumn[1]~Column = 3
aColumn[1]~Function = bsf.getConstant("com.sun.star.sheet.GeneralFunction","SUM")

/*apply SubTotals to range*/
xSubDesc~addNew(aColumn, 0)
xSub~applySubTotals(xSubDesc, .true)

/* insert  value into cells */
CALL UNO.setCell xSheet, 3, 2, 29

::requires UNO.cls      -- get UNO support
```

The result can be seen in Figure 20.

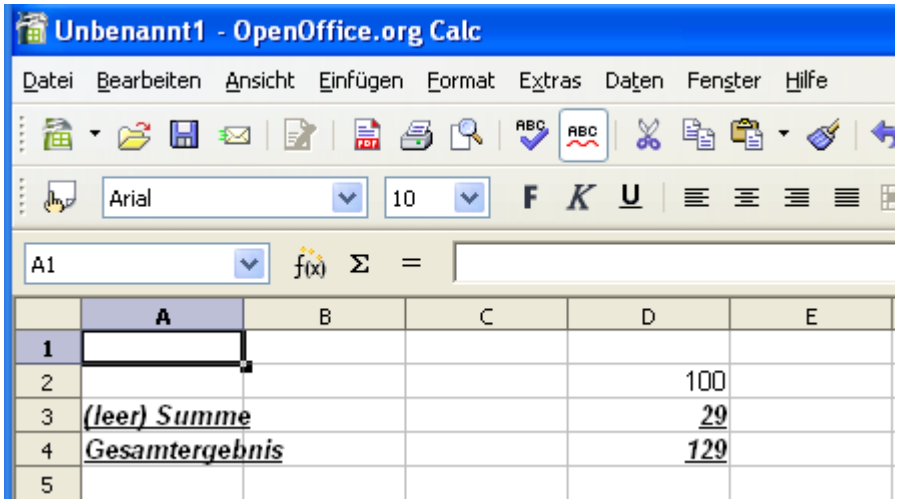


Figure 20: Subtotal

The lines of code explained in more detail:

By invoking *createSubTotalDescriptor(.true)* (Cutout.1) a SubTotalDescriptor object is created which is a method of the *XSubTotalCalculatable* interface.

Cutout.1

```
/*create SubTotalDescriptor( */
xSub = myRange~XSubTotalCalculatable
xSubDesc = xSub~createSubTotalDescriptor(.true)
```

Like in example08, Cutout.2 (p.47), the settings of the SubTotalDescriptor are set within an array and by calling the method *applySubTotals(xSubDesc, .true)* it is added to the sheet.

Cutout.2

```
/*apply SubTotals to range*/
xSubDesc~addNew(aColumn, 0)
xSub~applySubTotals(xSubDesc, .true)
```

7.2.12 Example12 - Annotation

This example adds an annotation to a cell with a given text.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)


/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

xCell= xSheet~getCellByPosition(1, 1)
xCellAddress = xCell~XCellAddressable~getCellAddress

CALL UNO.loadClass "com.sun.star.table.CellAddress"

/*get annotations*/
xAnnotation = xSheet~XSheetAnnotationsSupplier~getAnnotations
xAnnotation~insertNew(xCellAddress, "Rexx was here to make a annotation")

/*get annotations anchor and set visible*/
xAnnotAnchor = xCell~XSheetAnnotationAnchor
xAnnot =xAnnotAnchor~getAnnotation
xAnnot~setIsVisible(.true)


::requires UNO.cls       -- get UNO support
```

The result can be seen in Figure 21.

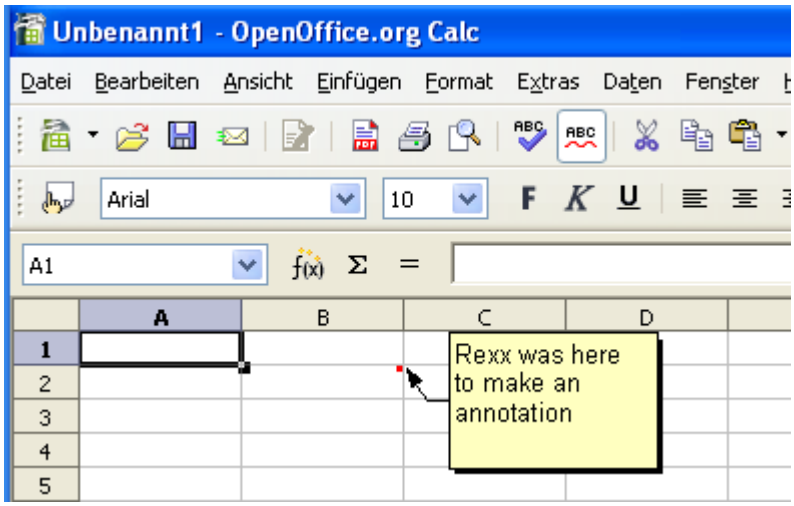


Figure 21: Annotation

The lines of code explained in more detail:

In Cutout.1 an annotation supplier is created and a new annotation is added to a cell.

```
Cutout.1
/*get annotations*/
xAnnotation = xSheet~XSheetAnnotationsSupplier~getAnnotations
xAnnotation~insertNew(xCellAddress, "Rexx was here to make a annotation")
```

The interface *XsheetAnnotationAnchor* and its method *setIsVisible()* are responsible for making the annotation visible.

```
Cutout.2
/*get annotations anchor and set visible*/
xAnnotAnchor = xCell~XSheetAnnotationAnchor
xAnnot =xAnnotAnchor~getAnnotation
xAnnot~setIsVisible(.true)
```

7.2.13 Example13 - Database

This example inserts data from existing a database into the active sheet. Note that the database has to be registered to OOo before executing this example. This is realized by clicking „New >> Database“. In the following window you have to choose „open a existing database“ and choose the wanted database.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)


/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/*set values for importing into an array*/
   importDesc = bsf.createArray(.UNO~propertyValue, 3)

importDesc[1] = .UNO~propertyValue~new
   importDesc[1]~Name  = "DatabaseName"
   importDesc[1]~Value = "rexx"

importDesc[2] = .UNO~propertyValue~new
   importDesc[2]~Name  = "SourceType"
   importDesc[2]~Value = bsf.getConstant("com.sun.star.sheet.DataImportMode","TABLE")

importDesc[3] = .UNO~propertyValue~new
   importDesc[3]~Name  = "SourceObject"
   importDesc[3]~Value = CD

xImport = xSheet~getCellRangeByName("A1:A1")
myImport = xImport~XImportable                          -- call interface XImportable
myImport~doImport(importDesc)                           -- import data

::requires UNO.CLS      -- get UNO support
```

The result can be seen in Figure 22.

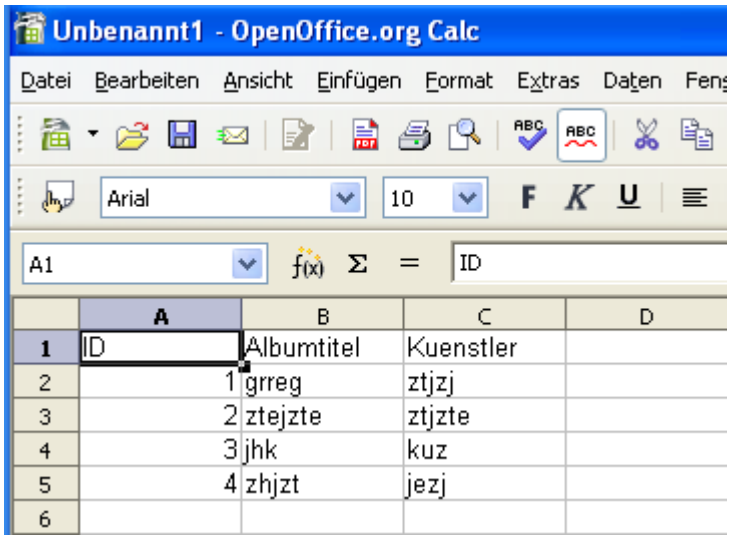


Figure 22: Database

The lines of code explained in more detail:

Cutout.1 shows which settings are necessary to import a table from a given database. Please note that the database, which is called „rexx“ in this example, has to be registered in OOo.

```
Cutout.1
importDesc[1] = .UNO~propertyValue~new
   importDesc[1]~Name  = "DatabaseName"
   importDesc[1]~Value = "rexx"

importDesc[2] = .UNO~propertyValue~new
   importDesc[2]~Name  = "SourceType"
   importDesc[2]~Value = bsf.getConstant("com.sun.star.sheet.DataImportMode","TABLE")

importDesc[3] = .UNO~propertyValue~new
   importDesc[3]~Name  = "SourceObject"
   importDesc[3]~Value = CD
```

It is not necessary to import a whole table because there are more possibilities to specify what you want to import:

- *If SourceType is TABLE, the whole table that is named by SourceObject is imported.*

- *If SourceType is QUERY, the SourceObject must be the name of a named query.*

- *If SourceType is SQL, the SourceObject is used as a literal SQL command string. [Deve05, p.651]*

The *XImportable* interface in Cutout.2 provides the method *doImport()* which is responsible for importing the data.

```
xImport = xSheet~getCellRangeByName("A1:A1")
myImport = xImport~XImportable                    -- call interface XImportable
myImport~doImport(importDesc)                     -- import data
```

7.2.14 Example14 - Scenario

This example adds a scenario to a cell range and names it „rexx“.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/*create cell range, get address and set into array*/
xCellRange = xSheet~xCellRange~getCellRangeByName("B3:F6")
xCellRangeAddress = xCellRange~XCellRangeAddressable~getRangeAddress
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
oAddr = bsf.createArray(.UNO~CellRangeAddress, 1)
oAddr[1] = xCellRangeAddress

aScenarioName = rexx
aScenarioComment = mein scenario

/* get  scenario and apply to cell range*/
xScenSupplier = xSheet~XScenariosSupplier
xScenarios = xScenSupplier~getScenarios
xScenarios~addNewByName(aScenarioName, oAddr, aScenarioComment)

::requires UNO.CLS       -- get UNO support
```

The result can be seen in Figure 23.

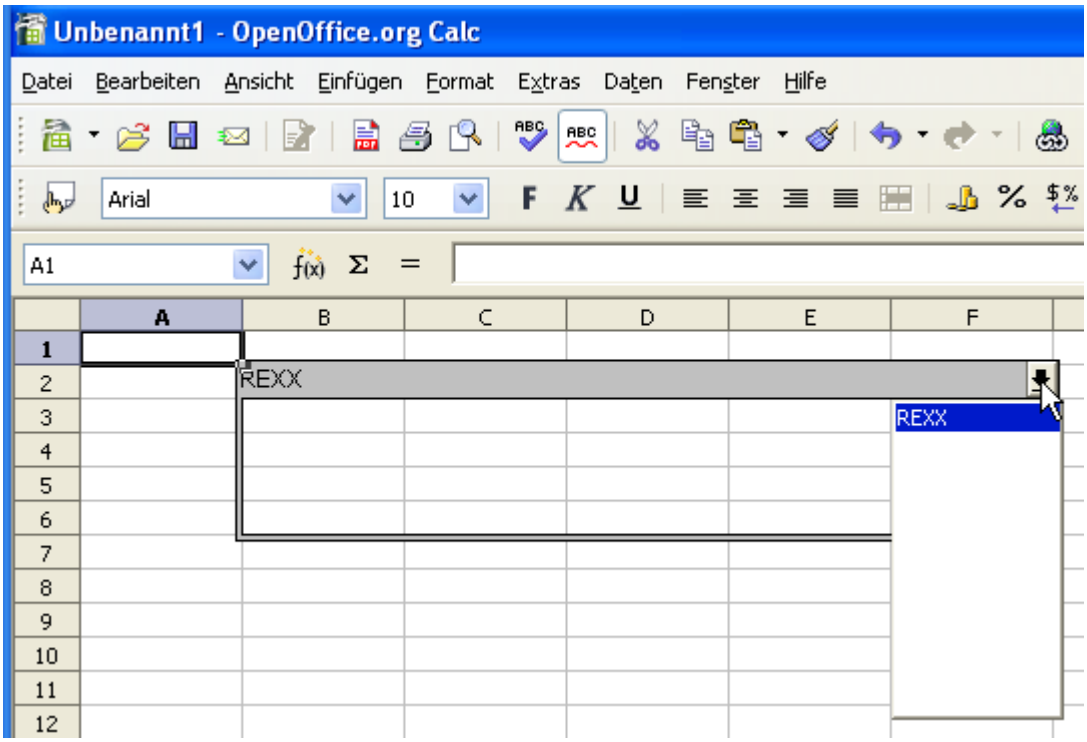


Figure 23: Scenario

The lines of code explained in more detail:

First a cell range is created (Cutout.1) where the scenario should be added. Then the cell range address is wrapped into an array. Therefore the class „*com.sun.star.table.CellRangeAddress*“ must be loaded by invoking the routine *UNO.loadClass()*.

Cutout.1

```
/*create cell range, get address and set into array*/
xCellRange = xSheet~xCellRange~getCellRangeByName("B3:F6")
xCellRangeAddress = xCellRange~XCellRangeAddressable~getRangeAddress
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
oAddr = bsf.createArray(.UNO~CellRangeAddress, 1)
oAddr[1] = xCellRangeAddress
```

In Cutout.2 the method *getScenarios* from the interface *XscenariosSupplier* is called to get the collection of scenario. Then a scenario is added using the *addNewByName()* method.

Cutout.2

```
/* get  scenario and apply to cell range*/
xScenSupplier = xSheet~XScenariosSupplier
xScenarios = xScenSupplier~getScenarios
xScenarios~addNewByName(aScenarioName, oAddr, aScenarioComment)
```

7.2.15 Example15 – Store

This example stores an OOo Calc document as a MS excel file.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  values into cells */
CALL UNO.setCell xSheet, 0, 0, "12"
CALL UNO.setCell xSheet, 0, 1, "43"
CALL UNO.setCell xSheet, 0, 2, "6"

xStorable = xDocument~XStorable                        -- call interface for storing

storeProps = bsf.createArray(.UNO~propertyValue, 1)

/*set values for storing into array*/
storeProps[1] = .UNO~propertyValue~new
storeProps[1]~Name = "FilterName"
storeProps[1]~Value = "MS Excel 97"

url1 = ConvertToURL(directory()"/oorexx.xls")
xStorable~storeAsURL(Url1, storeProps)



::requires UNO.CLS      -- get UNO support
```

The lines of code explained in more detail:

The interface Xstorable interface (Cutout.1) is responsible for storing files.

```
Cutout.1
xStorable = xDocument~XStorable                    -- call interface for storing
```

In Cutout.2 shows which values are needed to store the file as a Ms Excel File.

```
Cutout.2
storeProps = bsf.createArray(.UNO~propertyValue, 1)

/*set values for storing into array*/
storeProps[1] = .UNO~propertyValue~new
storeProps[1]~Name = "FilterName"
storeProps[1]~Value = "MS Excel 97"
```

In Cutout.3 ConvertToUrl specifies where the file should be stored and which extension it should have.

```
Cutout.3
url1 = ConvertToURL(directory()"/oorexx.xls")
xStorable~storeAsURL(Url1, storeProps)
```

7.2.16 Example16 - Split View

This example splits the view of a spreadsheet into four tiles.

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)



/* get first  and third sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet
xSheet1 = xDocument~getSheets~XIndexAccess~getByIndex(2)~XSpreadSheet

/*get controller from the Xmodel to manipulate view*/
xController = xDocument~XModel~getCurrentController

CALL syssleep 5

/*focus on new sheet and split view*/
xSpreadsheetView = xController~xSpreadsheetView~setActiveSheet(xSheet1)
xSpreadsheetsplit = xController~XViewSplitable~splitAtPosition(400, 200)

::requires UNO.CLS                  -- get UNO support
```

The result can be seen in Figure 24.

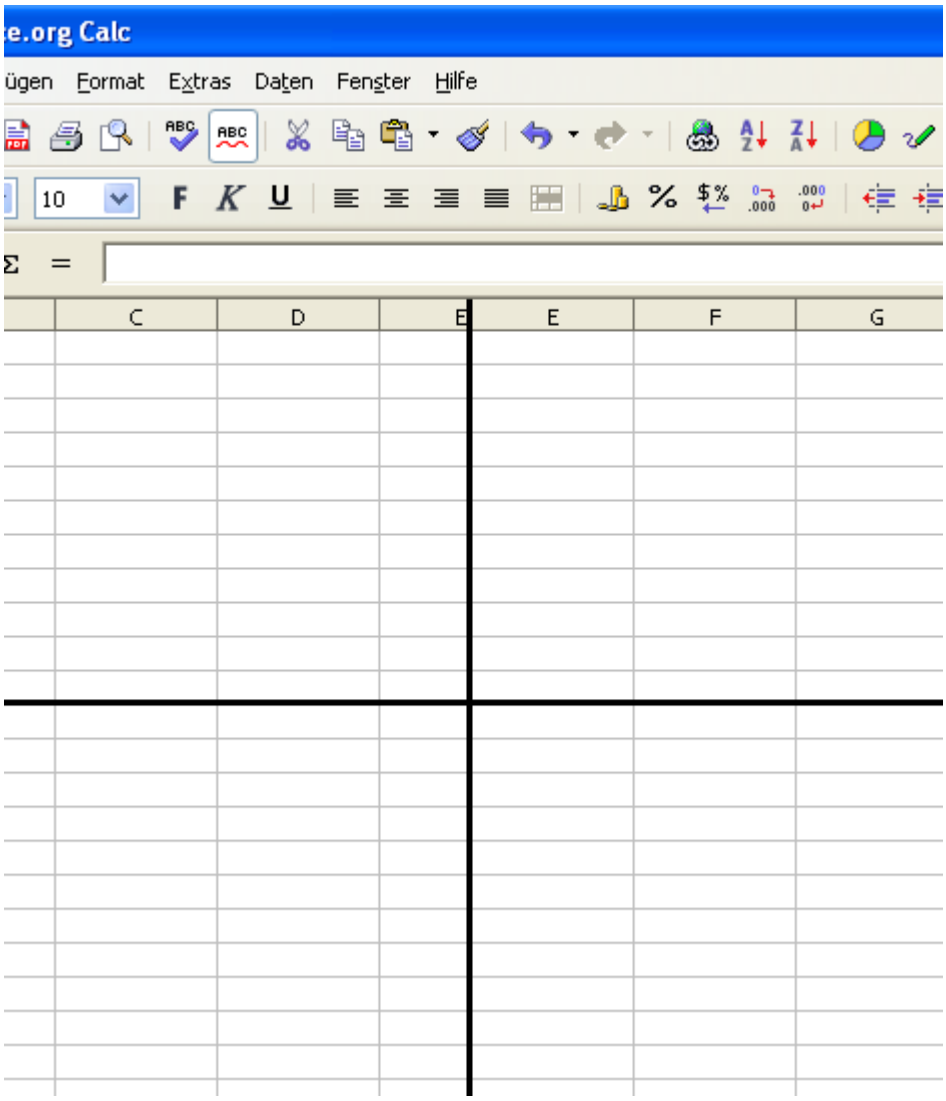


Figure 24: Split View

Like in example 3 (p.36) the controller of the Xmodel is needed to manipulate the view. The interface XviewSplitable interface with its method *splitAtPosition()* is responsible to split the view. The values of *splitAtPosition()* are pixel positions.

```
Cutout.1
xSpreadsheetsplit = xController~XViewSplitable~splitAtPosition(400, 200)
```

7.2.17 Example17 – Datapilot

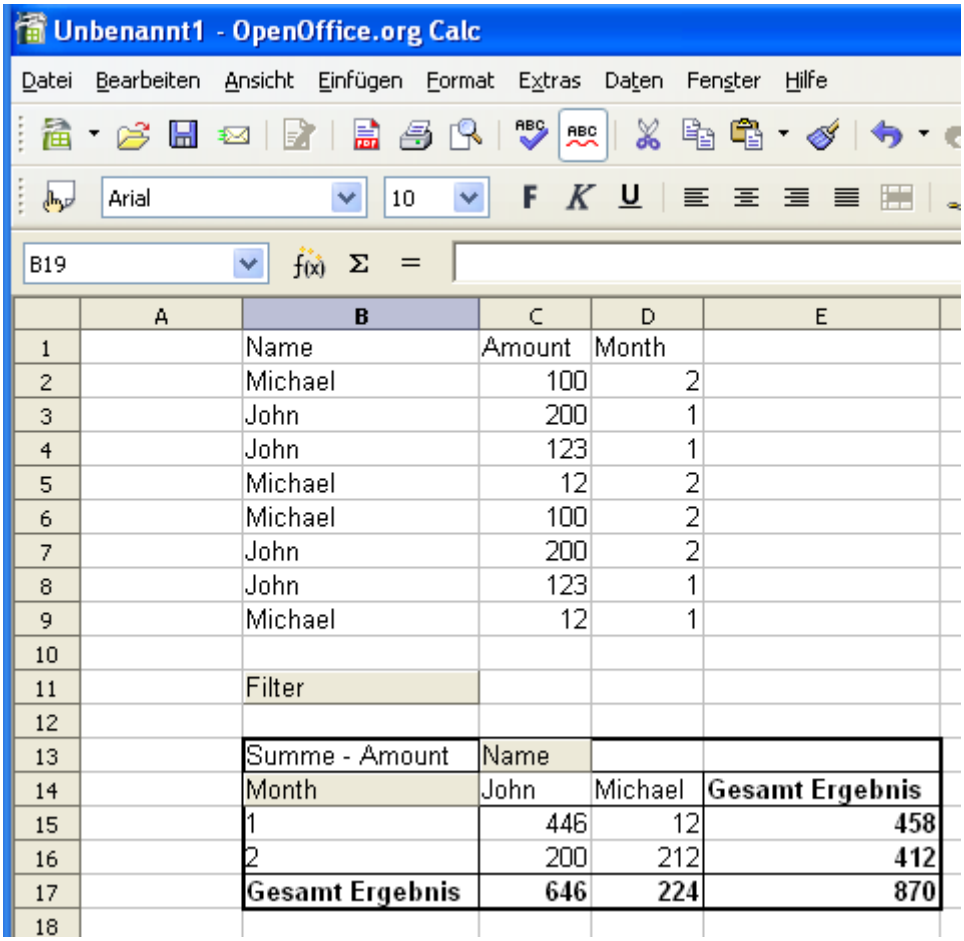
This example adds a datapilot to the spreadsheet.

> *The DataPilot feature in OpenOffice.org API Calc makes use of an external component that provides the tabular results in the DataPilot table using the field orientations and other settings that are made in the DataPilot dialog or interactively by dragging the fields in the spreadsheet. Such a component might, for example, connect to an OLAP server, allowing the use of a DataPilot table to interactively display results from that server.[Deve05, p.660]*

```
/* get the desktop  (an Xdesktop object) */
oDesktop = UNO.createDesktop()
xComponentLoader = oDesktop~XDesktop~XComponentLoader      --get componentLoader interface

/* open a blank calc file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)
/* get first  sheet in spreadsheet */
xDocument = xCalcComponent~XSpreadSheetDocument
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert  values into cells */
CALL UNO.setCell xSheet, 1, 0, "Name"
CALL UNO.setCell xSheet, 2, 0, "Amount"
CALL UNO.setCell xSheet, 3, 0, "Month"
CALL UNO.setCell xSheet, 1, 1, "Michael"
CALL UNO.setCell xSheet, 1, 2, "John"
CALL UNO.setCell xSheet, 1, 3, "John"
CALL UNO.setCell xSheet, 1, 4, "Michael"
CALL UNO.setCell xSheet, 1, 5, "Michael"
CALL UNO.setCell xSheet, 1, 6, "John"
CALL UNO.setCell xSheet, 1, 7, "John"
CALL UNO.setCell xSheet, 1, 8, "Michael"
CALL UNO.setCell xSheet, 2, 1, 100
CALL UNO.setCell xSheet, 2, 2, 200
CALL UNO.setCell xSheet, 2, 3, 123
CALL UNO.setCell xSheet, 2, 4, 12
CALL UNO.setCell xSheet, 2, 5, 100
CALL UNO.setCell xSheet, 2, 6, 200
CALL UNO.setCell xSheet, 2, 7, 123
CALL UNO.setCell xSheet, 2, 8, 12
CALL UNO.setCell xSheet, 3, 4, 2
CALL UNO.setCell xSheet, 3, 1, 2
CALL UNO.setCell xSheet, 3, 2, 1
```

```
CALL UNO.setCell xSheet, 3, 3, 1
CALL UNO.setCell xSheet, 3, 5, 2
CALL UNO.setCell xSheet, 3, 6, 2
CALL UNO.setCell xSheet, 3, 7, 1
CALL UNO.setCell xSheet, 3, 8, 1

/*get  cell range*/
xCellRange = xSheet~xCellRange~getCellRangeByName("A1:D9")
xCellRangeAddress = xCellRange~XCellRangeAddressable~getRangeAddress

/*create createDataPilotDescriptor by calling  XDataPilotTablesSupplie*/
xDataSupplier = xSheet~XDataPilotTablesSupplier
xData = xDataSupplier~getDataPilotTables()
xDataDescript = xData~createDataPilotDescriptor()

xDataDescript~setSourceRange(xCellRangeAddress)

myRange = xSheet~getCellByPosition(1, 10)
myAddr = myRange~XCellAddressable~getCellAddress

xFields = xDataDescript~getDataPilotFields()     -- get  DataPilotFields

/*apply values to DataPilotFields*/
aFieldObj = xFields~getByIndex(1)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","COLUMN"))
aFieldObj = xFields~getByIndex(3)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","ROW"))
aFieldObj = xFields~getByIndex(2)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","DATA"))

aFieldObj~xPropertySet~setPropertyValue("Function", -
bsf.getConstant("com.sun.star.sheet.GeneralFunction", "SUM"))

/*apply data pilot  to sheet*/
xData~insertNewByName("DataPilotExample", myAddr, xDataDescript)

::requires UNO.CLS   -- get UNO support
```

The result can be seen in Figure 25.



	A	B	C	D	E
1		Name	Amount	Month	
2		Michael	100	2	
3		John	200	1	
4		John	123	1	
5		Michael	12	2	
6		Michael	100	2	
7		John	200	2	
8		John	123	1	
9		Michael	12	1	
10					
11		Filter			
12					
13		Summe - Amount	Name		
14		Month	John	Michael	**Gesamt Ergebnis**
15		1	446	12	**458**
16		2	200	212	**412**
17		**Gesamt Ergebnis**	**646**	**224**	**870**
18					

Figure 25: Datapilot

The lines of code explained in more detail:

First it is necessary to create a DataPilotDescriptor which is shown in Cutout.1.

Cutout.1

```
/*create createDataPilotDescriptor by calling  XDataPilotTablesSupplie*/
xDataSupplier = xSheet~XDataPilotTablesSupplier
xData = xDataSupplier~getDataPilotTables()
xDataDescript = xData~createDataPilotDescriptor()
```

Then the DataPilotFields (Cutout.2) have to be retrieved to apply a value to it. First the first column is added as the column field. Then the third column is used as row field and the second colum is set as data field. Afterwards the function, which is specified as calculating the sum is set.

```
Cutout.2
xFields = xDataDescript~getDataPilotFields()     -- get  DataPilotFields

/*apply values to DataPilotFields*/
aFieldObj = xFields~getByIndex(1)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","COLUMN"))
aFieldObj = xFields~getByIndex(3)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","ROW"))
aFieldObj = xFields~getByIndex(2)
aFieldObj~xPropertySet~setPropertyValue("Orientation", -
bsf.getConstant("com.sun.star.sheet.DataPilotFieldOrientation","DATA"))

aFieldObj~xPropertySet~setPropertyValue("Function", -
bsf.getConstant("com.sun.star.sheet.GeneralFunction", "SUM"))
```

Cutout.3 shows how the data pilot is added to the sheet.

```
Cutout.3
/*apply data pilot  to sheet*/
xData~insertNewByName("DataPilotExample", myAddr, xDataDescript)
```

88 Conclusion

As the paper shows, the BSF4Rexx is able to build a powerful bridge between the

programming language Java and the scripting language ooRexx. Therefore it is possible to automate a whole office suite like OpenOffice.org.
By combining the components that were described in the first and second part of this work, which are all free of charge, it is possible to automate procedures of daily working life.
The nutshell examples showed that the automation of OOo can be used to a wide range of scenarios. But there are even more capabilities because this paper has only focused on one component of the office suite.

Due to the fact that there exist only a few examples dealing with OOo and ooRexx, it isn't always easy to figure out which interfaces are needed and which methods are available to obtain a specific functionality. But studying the OOo's Developers Guide [Deve05] and the OOo Api will be very helpful [ApiO06].

9 References

[ApiO06] OpenOffice.org Api Project.
http://api.openoffice.org/, , retrieved on 2006-06-25

[Burg06] Burger, Martin: OpenOffice.org Automation with Object Rexx, Bachelor Course Paper, 2005, Wirtschaftsuniversität Wien (Vienna University of Economics and Business Administration), Austria http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2006/200605_Burger/Bakk_Arbeit_Burger20060519.pdf, retrieved on 2006-06-21

[BSF406] BSF4Rexx Module:changesBSF4Rexx.txt, 2006 http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/changesOOo.txt, retrieved on 2006-06-02

[BSF406-1] BSF4Rexx Module: readmeBSF4Rexx.txt, 2006 http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/readmeBSF4Rexx.txt, retrieved on 2006-06-02

[Code06] OO-Snippets: Main Page http://codesnippets.services.openoffice.org/, retrieved on 2006-06-25

[Deve05] Sun Microsystems, Inc.:OpenOffice.org 2.0 Developers Guide, 2005

[Flat05] Flatscher, Rony G., Automatisierung von WindowsAnwendungen (4) – Abstrakter Datentyp, Klassen, Methoden, Attribute, Nachrichten, Geltungsbereiche, Generalisierungshierarchie, Vererbung http://wi.wu-wien.ac.at/rgf/wu/lehre/autowin/material/folien/Automatisierung_04.pdf, retrieved on 2006-04-11

[Flat05-1] 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/wu/lehre/autowin/material/folien/Automatisierung_04.pdf,

retrieved on 2006-04-11

[Flat06] 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 2006-06-17

[Flat06-1] Flatscher Rony G., Automatisierung von WindowsAnwendungen (1) – Einführung, Überblick, Anweisungen, Prozeduren, Funktionen. http://wi.wu-wien.ac.at/rgf/wu/lehre/autowin/material/folien/Automatisierung_01.pdf, retrieved on 2006-04-11

[Flat06-2] 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 2006-04-11

[Flat06-3] Flatscher, Rony G.: The Vienna Version of BSF4Rexx, 2006, Presentation at the 2006 International Rexx Symposium, USA http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf, retrieved on 2006-06-17

[Hahn05] Hahnekamp, Rainer: Extending The Scripting Abilities Of OpenOffice.org With BSF And JSR-223, Bachelor Course Paper, 2005, Wirtschaftsuniversität Wien (Vienna University of Economics and Business Administration), Austria

http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2005/200502_OOo-Hahnekamp/200502_Hahnekamp.pdf, retrieved on 2006-04-03

[Open06] OpenOffice.org: OpenOffice.org 2 - Product Description http://www.openoffice.org/product/index.html, retrieved on 2006-05-05

[Oore06] Rexx Language Association: About Open Object Rexx. http://www.ooRexx.org/, retrieved on 2006-04-12

[Wiki06] Wikimedia Foundation, Inc.: OpenOffice.org. http://de.wikipedia.org/wiki/Openoffice