OpenOffice.org Automation: Object Model, Scripting Languages, "Nutshell"-Examples

Andreas Ahammer
Vienna University of Economics and Business Administration,
Reg. No. 0251406

E-Mail: h0251406@wu-wien.ac.at

Intial version: June 24, 2005
Actual version: November 06, 2005

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

Contents

Figures

1 Introduction

1.1 About this paper

This paper is about the automation of OpenOffice.org via ObjectRexx. At first you'll get a short introduction about office applications and automation of office applications. It's followed by the overall concept, the technical requirements (OpenOffice.org, Java, BSF, ObjectRexx) and the object model. The next point is a short guidance how to install all the components so that you can use the presented “nutshells” to develop your own examples. The main part of this paper is the “Nutshell-Examples” where you can see how the automation of OpenOffice.org works.

1.2 Research question

How is it possible to automate the applications of OpenOffice.org by ObjectRexx?

1.3 Office Applications

One of the most frequently used software products is the so called “office” applications. People who work with the computer need such office applications at work and at home to write articles, memos and letters, make some calculations, create a presentation, draw a shape, and so on. So a lot of software companies and organizations offer office applications in a very wide variety.

On the one hand there are the commercial products where you have to buy a license to use it. The most famous commercial office application is Microsof's „Microsoft Office“ (MS-Office). MS-Office is like all other office applications a typical end-user tool[1], because it's easy to install, easy to use, it contains all necessary categories, all applications have a similar graphical user interface and it is „quasi standard“[2] to share documents. The disadvantages of MS-Office are, that it is very expensive (you have to pay for every new version), it only runs under „Microsoft Windows“ and the object model[3] is not really object oriented[4].

---

1 An end-user is a user of an application program. Typically, the term means that the person is not a computer programmer. It's a person who uses a computer as part of their daily life or daily work, but is not interested in computers per se. So an end-user tool is an application like an office software that the end-user uses.

2 Quasi standard means, when sharing a document (e.g. a text document, spreadsheet document) many users are going to take the propriate file – format of this application.

3 An object model is a specification of the objects for a given system, including a description of the object characteristics and a description of the static and dynamic relationships that exist between objects.

4 Object-oriented programming (OOP) is a programming language model organized around "objects". Programmers can define a data type, data structure and types of operations that can be applied to the data structure. So an object includes data and functions.

On the other hand there are a lot of open source[5] solutions like OpenOffice.org (OOo). The advantage of the open source components are that they have nearly the same features like the commercial products, you'll find a lot of information about them on the internet and, of course, they are free.

1.4 Automation

Automation in our context means to control the office application (OpenOffice.org) via a scripting language[6] (ObjectRexx).

Such a code to automate a program is often called a "macro".

> *"Macros support commands that allow a variety of advanced functions, such as making decisions, looping and even interacting with a person. ... Macros are especially useful when you have to do a task the same way over and over again, or when you want to press a single button to do something that normally takes several steps." [Pito04]*

So macros should facilitate the „every-day" work and therefore it is also an economical factor. If an employee doesn't spend a lot of time to do trivial tasks over and over again, the company will save money. For example would it be nice if you have to click only once with the mouse and an inquiry will convert to an order.

OpenOffice.org provides a very large variety for automation:

- Macro Recorder

  This is the easiest way to write a macro. It's possible to start the macro recorder from every OOo application and it stores every click with the mouse and every key press internal in a scripting language called "OpenOffice.org Basic". The advantage of the macro recorder is, that it is very easy to use and so nearly everyone can write a macro, but it has limited functions.

- OpenOffice.org Basic (OOo Basic)

  OOo Basic was developed for OpenOffice.org and *is a very flexible macro language, allowing automation of both simple and complex tasks*. [Pito04] The programmer himself writes line per line of the code and so he needs a lot of programming knowledge and has to know the Syntax of OOo Basic. Although this language is easy to use, has a lot of implemented features and is very powerful for scripting OOo, it has also some disadvantages: it doesn't have many features like other programming languages (e.g. Java – packages, API,...) and you can't communicate with other languages.

---

[5] Generically, open source refers to a program in which the source code is available to the general public for use and/or modification. Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community.

[6] A scripting language is a high level programming language that is interpreted by another program at runtime. Scripts typically interact either with other programs or with a set of functions provided by the interpreter. There are many scripting languages which can be embedded within HTML to add more functionality to a web page, such as different menu styles or interative formulars.

- Java

  OpenOffice.org offers a Java programming interface called “Java UNO“ (cp. *3.1 Universal Network Object*). So programmers which can handle Java well are able to automate OpenOffice.org. Java has some other advantages like a lot of implemented packages, the Java API and a very large community, but it is never as easy to learn as scripting languages. It takes time to understand the syntax of Java and to get a survey of how does Java work.

- ObjectRexx

  ObjectRexx is a powerful and easy to learn scripting language and is described in detail in *2.4 ObjectRexx*. By means of BSF4Rexx, ObjectRexx is able to control any Java class and so you can use ObjectRexx for OpenOffice.org automation.

2 Technical Requirements

This chapter introduces the technical requirements you'll need to automate OpenOffice.org from ObjectRexx.

2.1 Overall concept

The following figure shows the overall concept of the automation:

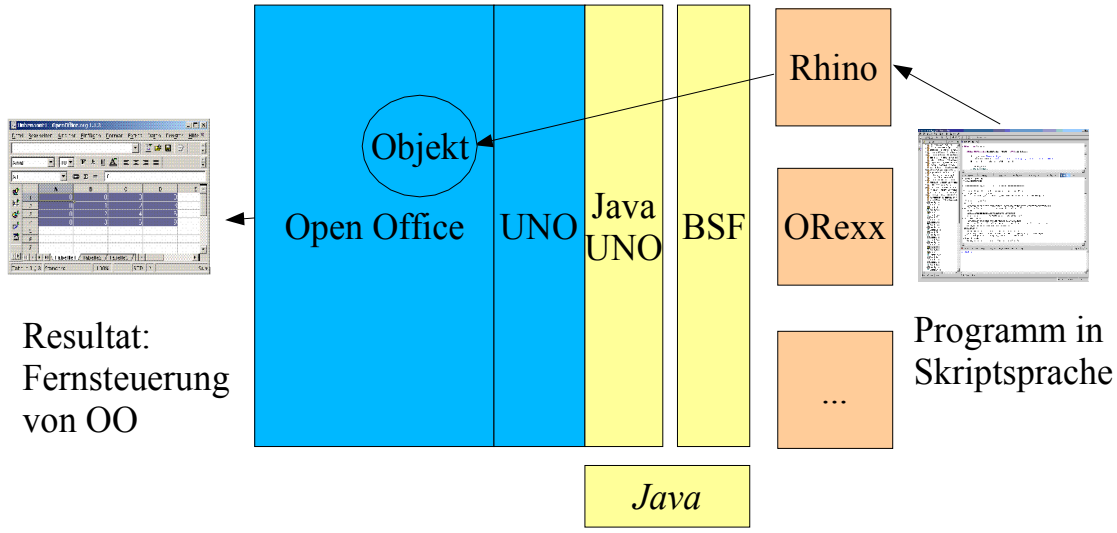

Figure 01: overall concept, [Augu05]

On the right side of *Figure 01* there is a scripting language which is compatible to the Bean Scripting Framework (BSF). The figure points out Rhino[7], because Walter Augustin describes in his paper [Augu05] some automation examples with this scripting language. Actually there could be any scripting language that's compatible to BSF.

The next part of the *Figure 01* is BSF and could be described as the bridge between a scripting language and Java. So it's going to convert the code from a scripting language to a Java-based code. BSF will be described in detail in *2.3 Bean Scripting Framework*.

The next part is called „Universal Network Object“ (UNO). UNO, in our case the Java UNO, provides a bridge between OpenOffice.org and a programming language (e.g. Java) to have access to all OpenOffice.org objects. Java UNO will be described in detail in chapter *3.1 Universal Network Object*.

On the left side there is OpenOffice.org with all its components. It is very important to be familar with the object model, i.e. how to get objects from OpenOffice.org and how to deal with them. The object model will be described in detail in chapter *3.2 Object Model of OpenOffice.org*.

[7] Rhino is an open-source implementation of JavaScript written entirely in Java. It is typically embedded into Java applications to provide scripting to end users. It can be downloaded at http://www.mozilla.org/rhino/.

2.2 OpenOffice.org

OpenOffice.org is an advanced and open source version of Sun's[8] StarOffice[9] and offers a lot of different office applications.

2.2.1 History

In 1984 a company called „Star Division“ was founded by Marco Börries. The main product was an office suite called „Star Office“. In 1999 Star Division was taken over by „Sun Microsystems“ and Sun made the software available to be downloaded for free. Sun has developed a new version of Star Office named „OpenOffice“. On July 19th, 2000 OpenOffice.org was released by Sun and on October 13th, 2000 the OpenOffice.org website was going online. So OpenOffice.org was born and now everyone can download the product and the sourcecode for free.
[at-mix05]

Since then over 16 million people downloaded OpenOffice.org and a large community was formed.
[Open05]

The program has been developed continously and the latest released version (November 2005) to download is OpenOffice.org 2.0 (http://de.openoffice.org/downloads/quick.html).

2.2.2 Applications

The following applications are provided by OpenOffice.org 1.1.4:

- swriter[10]

  swriter is a tool to create professional documents, memos, newsletters, webpages and booklets. It offers a great variety of formatting text, insert graphics, tables, diagrams and different styles.

- scalc

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

- sdraw

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

---

8 Sun is the short name for „Sun Microsystems“, a software company and, beside other very remarkable products, the developer of Java. For further informations look at http://www.sun.com/

9 Star Office was a commercial office application of „Star Division“ and the forerunner of OpenOffice.org

10 The „s“ in swriter comes from the „Star“ of StarOffice (the forerunner of OpenOffice). You'll find this „s“ in ervery OpenOffice.org document: swriter, scalc, sdraw and simpress

- simpress

  simpress is based on sdraw, but you can also make nice presentations.

- database module

  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 ODBC[11] or JDBC[12].

2.2.3 OpenOffice.org 2.0

At the moment (November 2005), OpenOffice.org 2.0 is already released. All examples were still developed with OpenOffice.org 1.1.4 and tested with OpenOffice.org 1.1.4 and OpenOffice.org 2.0 beta. If there are any differences between the two versions it will be pointed out directly at the example.

2.3 Bean Scripting Framework

The Bean Scripting Framework (BSF) is an open source project from IBM and is defined as followed:

> *„The Bean Scripting Framework is a set of Java classes that enables the use of scripting languages, such as Javascript and Tcl, within Java applications and that permits the use of Java objects and functions within supported scripting languages.“ [Apac02]*

The main functions of BSF are:

- embedding script code into a Java - based program environment
- use of Java components in scripting languages

[Hane05]

2.3.1 History

BSF was developed in 1999 by IBM. The initial intend was to provide access to JavaBeans from scripting languages environment. The significant development was done by Matt Dufler and Sam Ruby and BSF was incorporated into Websphere (an IBM product) and Xalan (an Apache project). Today BSF is a part of the Apache Software Foundation from IBM.
[IBM05]

---

[11] Open Database Connectivity, in short ODBC, is a standard database access method developed by the SQL access group. The goal of ODBC is to make it possible to access any data of the datebase from any application.

[12] Java Database Connectivity, in short JDBC, is the Java version of ODBC. It's an Java API that provides Database Management System connectivity to a wide range of SQL databases.

2.3.2 Architectural Overview

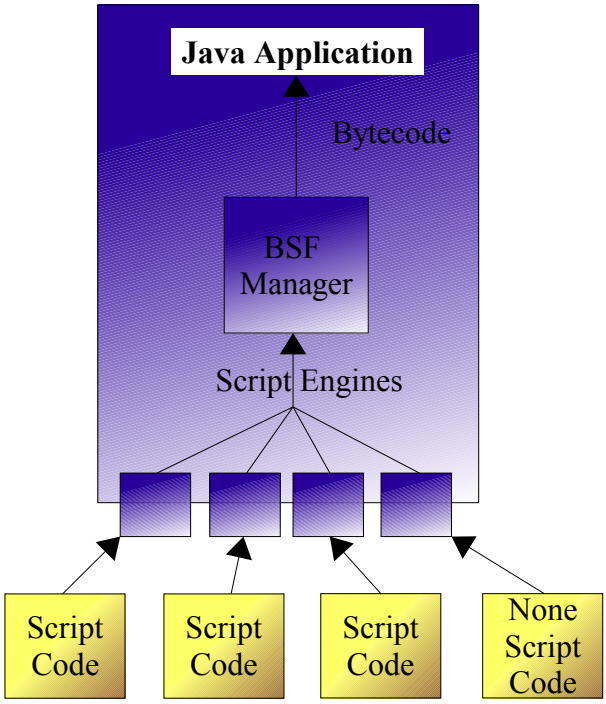

Figure 02: BSF – architectural overview, [Hane05]

As the figure above shows, the primary components of BSF are the BSFManager and the BSFEngine.

> *„The BSFManager handles all scripting execution engines running under its control, and maintains the object registry that permits scripts access to Java objects.“ [Apac02]*

> *„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.“ [Apac02]*

2.3.3 BSF4Rexx

As the name already assumes, BSF4Rexx is the Bean Scripting Framework for ObjectRexx. BSF4Rexx was developed in 2000 by Prof. Flatscher and his student Peter Kalender at the university of Essen. In the „Essener version“ it was possible for Java based programs to call a Rexx based program. That means Java developer could use ObjectRexx as scripting language.

In 2003 an advanced version of BSF4Rexx released and was called „Augsburger version“. Now it was possible to handle Java classes from ObjectRexx. Every Java class can be used from ObjectRexx, so that the Java API is like a huge library for ObjectRexx.

The newest version of BSF4Rexx is the „Wiener version“ and there is an ongoing development. The key feature of the current version is, that no strict typing is required and a lot of new functions for OpenOffice.org automation were created.
[Flat05]

The newest version of BSF4Rexx can be downloaded at
http://wi.wu-wien.ac.at/rgf/rexx/BSF4Rexx/ .

The following figure shows the BSF4Rexx architecture of the „Wiener version“:

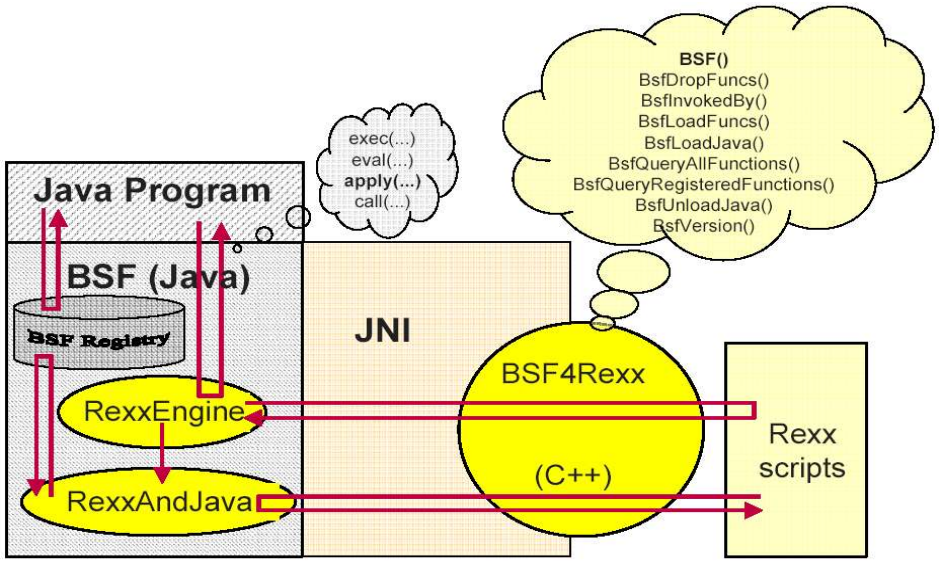

Figure 03: BSF4Rexx – architectural overview of the „Wiener version“, [Flat05]

Because OpenOffice.org provides a Java interface (the JavaUNO) and ObjectRexx can be interpreted to Java via BSF4Rexx, ObjectRexx can be used for OpenOffice.org automation.

2.4 ObjectRexx

With the words of the developer Mike F. Cowlishaw is Rexx *„a procedural language that allows programs and algorithms to be written in a clear and structured way."*
[Mart05]

2.4.1 History

Restructured Extended Executor (Rexx) is a follower of the scripting language EXEC and was developed in 1979 by Mike F. Cowlishaw, an employee and now a Fellow[13] of IBM. Since 1987 Rexx was available for all IBM platforms and later more ore less for all operating systems.

[13] An IBM – Fellow is an appointed position at IBM. It is considered to be the highest honor a technologist at IBM can achieve.

In the early 1990s the IBM user-association initiated the development of an object oriented version of Rexx called ObjectRexx. ObjectRexx is completely compatible to Rexx but has object oriented features and a powerful object model.

In 1996 a new version of Rexx, NetRexx was designed. NetRexx looks like Rexx and has essentially the same syntax, but internally it generates a Java code.
[Flat05]

The newest version of Rexx is OpenObjectRexx and is a open source project by the RexxLA[14], providing a free implementation of ObjectRexx (http://www.oorexx.com/).

2.4.2 Features

The main goal of the developer was to create a powerful, full-featured programming language which has an human-oriented syntax and is easy to learn and easy to use.

The following aspects are the key features of Rexx:

- an english-like language

  To make Rexx easy to learn and easy to use many instructions are meaningful english words like `SAY, IF ... THEN, DO ... END, DO ... WHILE, EXIT,...`

- fewer rules

  Rexx has no strict rules about formatting the source code. One single command can span many lines and also in one line there can be several commands. Instructions are not case-sensitive, i.e. you can type them in upper-case, lower-case or mixed-case. When a Rexx-program is running, internally all characters are convert to upper-case (excepting special characters, numbers and strings under apostrophe). Instructions can be finished with a semicolon, but it is not necessary.

- interpreted, not compiled

  Rexx is an interpreted language. When a Rexx program executes, the language processor reads each statement from the source file and runs it.

- built-in functions and methods

  There are many built-in functions and methods that perform various processing, searching and operating functions for text and numbers. Available functions are described in [Flat05] and [Muel01].

- typeless variables

  The next wide difference to other programming languages like Java is, that Rexx has no strict type definition. Variables can hold any type of object, so you don't

[14] The Rexx Language Association (RexxLA, http://www.rexxla.org/) is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language.

need to declare variables as strings or numbers. Rexx knows internally which datatype the variables are.

- clear error messages and powerful debugging

  Rexx displays messages with full and meaningful explanations when an error occurs.

[Chay05]

2.4.3 Syntax

Because all examples in this paper are written in ObjectRexx, here is a short introduction about the syntax of ObjectRexx. Note that this introduction is only made to make you capable to read and understand my examples. If you want to design your own nutshells study [Flat05] or [Muel01].

- comments

  There are 2 options to write a comment:

```
/* comment */     (this comment can span several lines)
-- comment        (this comment can span just one line)
```

- print something in the command line

```
SAY "Hello World"
SAY "Hello " || "World"  (combine 2 strings)
```

- variables

  There is no strict typing in ObjectRexx:

```
s = "Hello"
i = 12
```

- block

  A block starts with DO and ends with END:

```
DO
  instructions
END
```

- flow - control

```
IF i < 2 THEN DO
 instructions
END
```

There are many options to realize a loop:

```
i = 3
DO WHILE i < 3
  instructions
  i = i + 1
END
```

- procedures

There are some different possibilities to realize a procedure. In the examples below always a routine is used for a procedure.

```
::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)
```

The call of this routine looks like this:

```
call name 2, 3
```

- requires - directive

A requirement declares another Rexx program. It is the first directive which is called when a Rexx program is executed and so all public routines[15] of the other Rexx program are available. In the examples below there is always one requirement:

```
::requires UNO.cls      (to get OOo support)
```

- interaction with objects

Interaction with objects exclusively works with messages which are sent to the object. E.g. if you want to get the method „method1“ from the object „object1“:

```
object1~method1         (returns what the method1 returns)
```

The ~ („Twiddle“) is like the . in Java and returns what the method returns. To get as return statement the object itself you need ~~

```
object1~~method1    (returns object1)
```

- arrays

When automating OpenOffice.org a lot of arrays are used. An easy array could be created like this:

```
tmpColl = .array~new    (create a new array)
```

[15] When a routine is public, the routine can be called by another object with a message to the current object.

```
tmpColl[1] = "a"          (insert „a“ to the first position – ObjectRexx
                           starts with 1)
tmpColl~put("b", 2)       (another way to insert a variable)
```

- predefined functions

  ObjectRexx provides a lot of predefined functions like a „break“ where the program stops for x seconds:

```
call syssleep 2           (the program will stop for 2 seconds)
```

2.5 Operating System

As already mentioned above, OpenOffice.org, Java, BSF4Rexx and ObjectRexx are platform independent. So you can almost choose every operating system you want.

OpenOffice.org is available for Windows, Linux x86, Mac OSX (X11), PrOOo-Box, FreeBSD, Linux PowerPC, Linux Sparc, Solaris sparc and Solaris x86.
[Open05]

The standard edition of Java is available for Windows, Linux, Solaris sparc and Solaris x86.
[Sun05]

Object Rexx runs on every imaginable platform including Windows, Unix, Linux, BSD, Mac OS, IBM iSeries, OS/2, DOS and the major handheld operating systems.
[RexxLA05]

3 Concepts

3.1 Universal Network Object

> *„UNO (pronounced ['ju:nou]) stands for Universal Network Objects and is the base component technology for OpenOffice.org. You can utilize and write components that interact across languages, component technologies, computer platforms, and networks. Currently, UNO is available on Linux, Solaris, and Windows for Java, C++ and OpenOffice.org Basic. As well, UNO is available through the component technology Microsoft COM for many other languages. UNO is used to access OpenOffice.org, using its Application Programming Interface (API). The OpenOffice.org API is the comprehensive specification that describes the programmable features of OpenOffice.org.“ [Open03]*

That means UNO provides a powerful bridge between OpenOffice.org and programming languages.

The main Features of UNO are:

- different languages

  UNO can be implemented and accessed from any programming language where a UNO language binding exists. E.g. Walter Augustin made examples for Java, ObjectRexx, Rhino, VisualBasic, PHP, and OLE[16]. [Augu05]

- different operating systems

  OpenOffice.org can be downloaded for Windows, Linux and Solaris. So if the automation is done with a platform independent programming language you are not bound to a special operating system.

- different networks

  UNO is a client/server system. I.e. that a connection via TCP/IP[17] from a client to a server has to be initiated. The advantage is, that the client and the server could be on one computer, but it is also possible that the client is on the other side of the world than the server.

[Open03]

This 3 features make OpenOffice.org very flexible to automate without boundaries.

The special bridge that links Java to UNO is called JavaUNO.

---

[16] OLE (Object Linking and Embedding) is Microsoft's framework for a compound document technology.

[17] TCP/IP (Transmission Control Protocol / Internet Protocol) is a communication protocol for connecting computers through the internet.

3.2 Object Model of OpenOffice.org

One of the main question for OpenOffice.org automation is how to get objects and how to work with them.

The following figure shows which components are available to manage the object model:

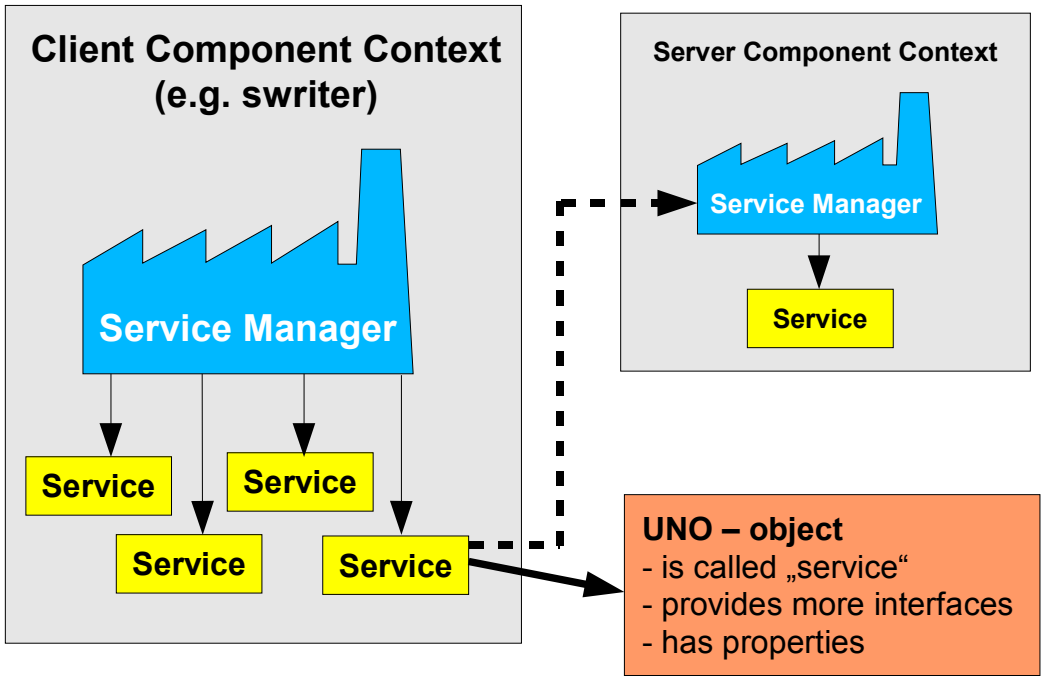

Figure 04: Service Manger, [Open03]

3.2.1 Service Manager

> *„UNO introduces the concept of service managers, which can be considered as factories that create services.“ [Open03]*

The service manager in *Figure04* is displayed as a factory, because it provides other services. So the service manager is the root component of all other objects and it has to be initiated in every OpenOffice.org automation. That's why it's also called the main factory.

For example, most of the examples below needs the `com.sun.star.frame.Desktop` service. The `Desktop` is used to load documents, to get the current document and to access all loaded documents.

Another service provided by the service manager is the `com.sun.star.sdb.Databasecontext`, which holds databases registered with OpenOffice.org.

The service manager initiates services by their name, to specify all implementations of a certain service and to add or remove factories for a certain service at runtime.

The main interface (for interfaces see *3.2.5 Interfaces*) of the service manager is the `com.sun.star.lang.XMultiComponentFactory`, which offers beside other methods the method `createInstanceWithContext()`. It returns a default constructed service instance which supports at least all interfaces which were specified for the requested service name.

When you are looking to older examples like [Augu05] you will see, that instead of `com.sun.star.lang.XMultiComponentFactory` the author uses the instance `com.sun.star.lang.XMultiServiceFactory`. `XMultiComponentFactory` is newer and should be used, but you can also still work with the `XMultiServiceFactory`.

Each component (swriter, scalc,...) has also it's own service manager to create contents within the component.
[Open03]

3.2.2 Component Context

> *„A service always exists in a component context, which consists of the service manager that created the service and other data to be used by the service“ [Open03]*

A component context could be an application of OpenOffice.org, e.g. swriter, scalc, and so on.

Often a component needs more functionality or information after the application is deployed. Therefore every component context has it's own service manager. If a table should be inserted to a swriter document, you need the `XmultiServiceFactory` within the component context.
[Open03]

3.2.3 How to get objects

> *„An object in our context is an instance of an implemented class that has methods you can call.“ [Open03]*

There are different objects available in OpenOffice.org:

- new objects

  New objects are created by the service manager. For example the instruction

```
xMultiComponentFactory~-
    createInstanceWithContext("com.sun.star.drawing.ShapeCollection", xContext)
```

  returns a new Desktop interface.

- document objects

  Document objects are the files that are opened with OpenOffice.org and are created with the `loadComponentFromURL()` method of the `Desktop` object.

- objects that are provided by other objects

  If an object is an integral part of another object, the parent object has often a get – method to get the object. For example the `xText` object is a part of the `xTextDocument` and so the `xTextDocument` provides a `getText()` method to get a `xText` object.

  ```
  xText = xTextDocument~getText
  ```

  Other features, called properties, which are not considered integral for the architecture of an object are accessible through a set of methods. For properties look at *3.2.6 Properties*.

[Open03]

3.2.4 Services

As aforesaid, all objects in the OpenOffice.org API are called services. But objects and services are usually not the same thing, because all UNO objects have to follow a service specification and have to support at least one service.

> *„Services describe objects by combining interfaces and properties into an abstract object specification.“ [Open03]*

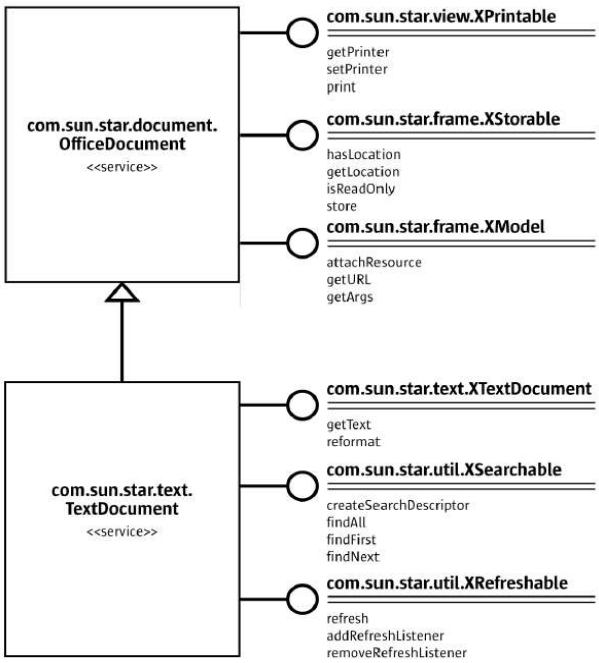

Figure 05: Service – Text Document, [Open03]

The figure above shows a cut-out of the service `com.sun.star.text.TextDocument` in UML notation.

On the left side of *Figure5* you can see, that the `com.sun.star.text.TextDocument` must include the service `com.sun.star.document.OfficeDocument` by definition.

On the right side there are all interfaces the service must support. The `TextDocument` has to support the `XTextDocument`, the `XSearchable` and the `XRefreshable` interface. Because it is also an `OfficeDocument`, it supports the interfaces `XPrintable`, `XStoreable`, `XModel` and `XModifiable` (which is not shown in *Figure5*) too.

The properties of the interface `XTextDocument` (`CharacterCount`, `ParagraphCount`, `WordCount` and `XPropertySet`) are not shown in *Figure5*. If you want an overview of all interfaces and properties of `XtextDocument`, visit the OpenOffice.org API: http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextDocument.html. [Open03]

3.2.5 Interfaces

> *„An interface is a set of methods that together define one single aspect of a service.“ [Open03]*

For example according to *Figure5*, the interface `com.sun.star.view.XPrintable` prescribes the methods `print()`, `getPrinter()` and `setPrinter()`.

An interface to use is not simple. When you are working with Java, the normal way is to cast an object before you can call a method from it. But UNO objects are quite different. When you want to access the methods of an interface, you must ask the UNO environment to get the appropriate reference. However, the compiler doesn't yet know about it. So the Java UNO environment has a `queryInterface()` method to cast all UNO types safe across process boundaries.
[Open03]

Because BSF4Rexx integrates the ObjectRexx code into a Java environment, these circumstances affects the ObjectRexx code.

If we want the `XSpradsheetDocument` interface from the `XCalcComponent`, the Java code may look like this:

```
XSpreadsheetDocument xSpreadsheetDocument = (XspreadsheetDocument)
      UnoRuntime.queryInterface( XSpreadsheetDocument.class,
      xCalcComponent);
```

The appropriate ObjectRexx code is:

```
xSpreadsheetDocumentName = .BSF4Rexx~Class.class~forName -
      ("com.sun.star.sheet.XSpreadsheetDocument")

xSpreadsheetDocument = unoRuntime~queryInterface -
      (xSpreadsheetDocumentName, xCalcComponent)
```

Note that the „-“ is because of the line break!

With thanks to Prof. Flatscher, who is still developing BSF4Rexx, the ObjectRexx code now looks like this:

```
xSpreadsheetDocument = xCalcComponent~XSpreadsheetDocument
```

This makes the code easier to write and easier to read.

3.2.6 Properties

> *„A property is a feature of a service which is not considered an integral or structural part of the service and therefore is handled through generic* `getPropertyValue()` / `setPropertyValue()` *methods instead of specialised get methods... .“ [Open03]*

An interface that supports the `com.sun.star.beans.XPropertySet` interface can handle properties and so many features can be set through a single call which improves the remote performance.

The next instruction shows how to set the color of an `XTextCursor` with the method `setPropertyValue()`.

```
xTextCursorProps = xTextCursor~XPropertySet
xTextCursorProps~setPropertyValue("CharColor", -
      box("int", "ff 00 ff"x ~c2d))
```

The properties of a print job can be set with an array of name/value variables.

```
printOpts = .bsf~createArray(.OOo~propertyValue, 1)
printOpts[1] = .bsf~new("com.sun.star.beans.PropertyValue")
printOpts[1]~bsf.setFieldValue("Name", "Pages")
printOpts[1]~bsf.setFieldValue("Value", "1")
```

4 Installation Guide

Because you have many software components to install and to configure for using the examples below, here is a short installation guide.

1. OpenOffice.org

   Download the newest version (at the moment in November 2005 OpenOffice.org 2.0) of OpenOffice.org from http://www.openoffice.org/ and install it on your computer. Then agree the license regulations and all the components of OpenOffice.org are ready to use.

   The next step is to make OpenOffice.org listen. Only when it is in listen-mode the connection from the programming language to OpenOffice.org is possible and so you can get all necessary UNO Objects. Regarding to [Open03] there are two options to set OpenOffice.org listen:

   1. An alternative is to launch the office in listening mode using command - line options. To do this, start it from the command - line:

      ```
      <OfficePath>/program/soffice "-accept=socket,port=8100;urp;"
      ```

      When you use this command - line option, the office will only listen during the current session.

   2. If you want to make OpenOffice.org always listen when the computer is turned on, edit the file "Setup.xcu" (`<OfficePath>/share/registry/data/org/openoffice`) and add following code in the "Office" - node (i.e. after the line: `<node oor:name="Office">`):

      ```
      <prop oor:name="ooSetupConnectionURL" oor:type="xs:string">
        <value>socket,host=localhost,port=8100;urp;</value>
      </prop>
      ```

      Don't forget to set your firewall and don't block port 8100 if you want to connect from an other computer.

   Now you have to set classpathes[18] to the required *.jar – files[19] of OpenOffice.org. The classpathes should be set to:

   ```
   <OfficePath>/program/classes/ridl.jar
   <OfficePath>/program/classes/juh.jar
   <OfficePath>/program/classes/unoil.jar
   <OfficePath>/program/classes/jurt.jar
   <OfficePath>/program/classes/sandbox.jar
   ```

[18] Classpath is a list of places where the operating system can find files that are required (e.g. Java classes)
[19] JAR is short for „Java Archive" which is a file format used to bundle all components required by a Java program.

2. Java

   The latest version of Java (at the moment in November 2005 Java 5.0) can be downloaded at http://java.sun.com/. Install the software and set a classpath to the bin-folder of your Java installation (`<JavaPath>/bin/`).

   If an example stops with an critical exception about the Java Virtual Machine (jvm), set a path to `<JavaPath>/jre/bin/client`.

3. ObjectRexx

   Since ObjectRexx was going open source the free version of ObjectRexx, called OpenObjectRexx, is available to download at http://www.oorexx.org/. All classpathes should be automatically set. If that's not the case, here are the classpathes which are required:

   ```
   <ObjectRexxPath>
   <ObjectRexxPath>OODIALOG
   ```

4. BSF4Rexx and UNO.cls

   There are many versions of BSF4rexx so you must be careful that you download the right one. The latest version you can get at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/. Download the files and extract it to a folder.

   Study the `readmeBSF4Rexx.txt` and the `readmeOOo.txt` to install both correctly.

Now you are ready to test the examples.

5 Examples

The „nutshell“ examples below demonstrate the OpenOffice.org automation. They show a lot of possibilities and should be understood as approach to create more programs with a higher complexity. It would be nice, if an employee has to click only once with the mouse button and OpenOffice.org provides a presentation of the available data that come from a server anywhere in the internet.

Some of the examples are taken from [Open03] and [Pito04] and were modified to the software environment. In this two books are a lot of examples which you can download at http://www.openoffice.org/dev_docs/source/sdk/ and http://www.pitonyak.org/oo.php.

If you want to test my examples you can still type them, use copy and paste or download them here: http://www.wu-wien.ac.at/usr/h02c/h0251406/e-commerce/vk6/.

The examples are divided into the 5 categories. The first 3 categories represent one of the main application of OpenOffice.org: swriter, scalc and sdraw and simpress. The 4th category is something about database functions and the last section presents a dictionary included in OpenOffice.org.

All the examples were developed on a WindowsXP computer with OpenOffice.org 1.1.4 and were also tested with OpenOffice.org 2.0 beta (more exactly with OpenOffice.org 1.9.79). If there are any differences or bugs you'll find it at the relevant example.

5.1 swriter – examples

Swriter offers a huge variety of functions to handle text. Such features are formatting the text, insert tables, text fields, bookmarks and footnotes, search and replace text, print pages, word counting, define styles, and so on. Other, maybe more interesting functions, are to insert text from another document, load a document from an URL[20] or export files directly to pdf[21].

Before we go to the classical „Hello World“ example some words about the `TextDocumentModel` of swriter:

[20] An Uniform Ressource Locator (URL) is an universal address for resources on the internet.
[21] A Portable Document Format (pdf) is a file format to transfer (text) files over the internet.

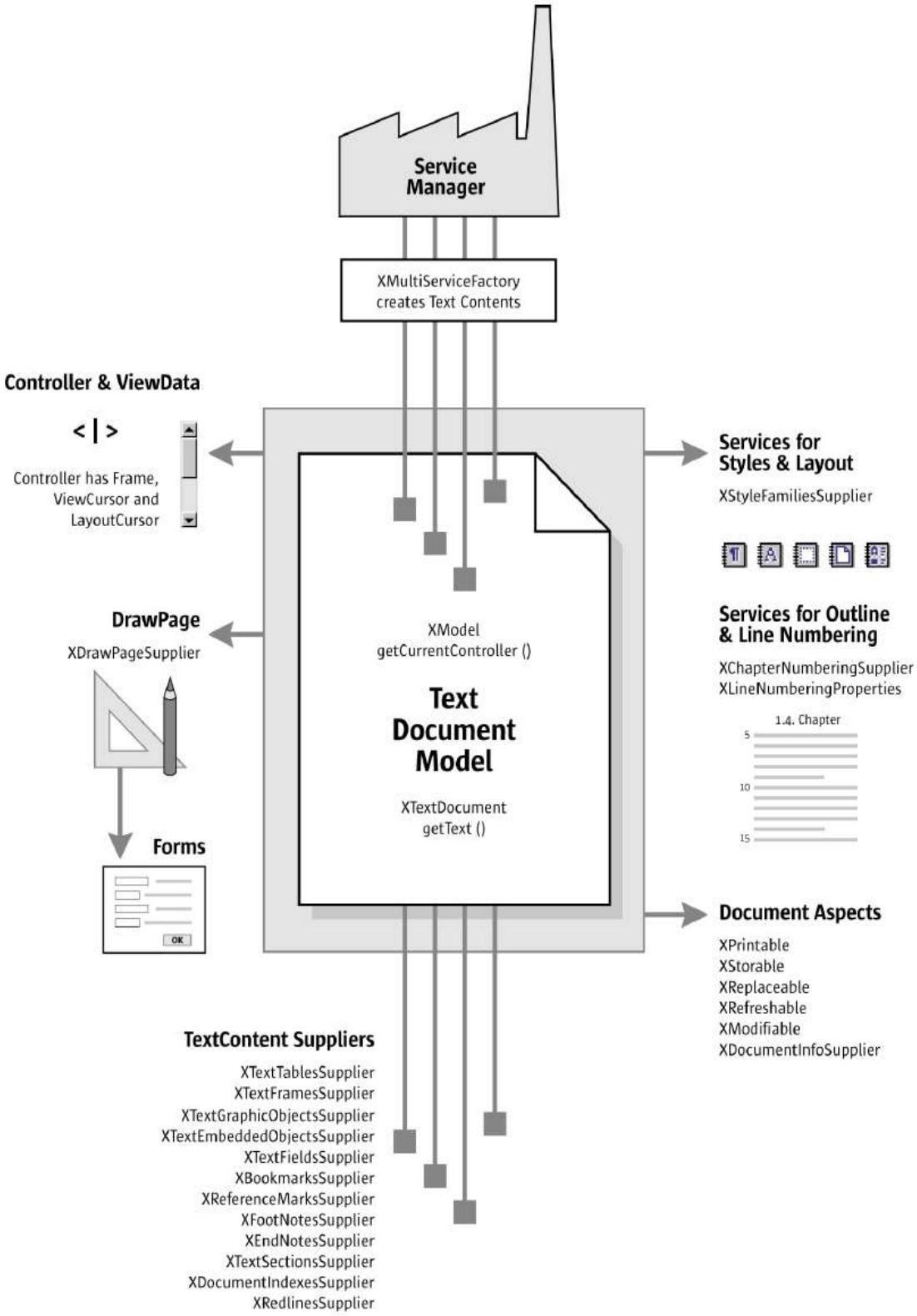

Figure 06: Text Document Model, [Open03]

The 5 major architectural areas are:

- text

  The text is the main part of every text document and is provided by the text document model. The text is responsible for the style of the text, organizes the paragraphs and knows about the text content.

- draw page

  The draw page is a transparent layer that lies above the text. So it's possible to draw shapes and to wrap the objects to the text.

- service manager

  The service manager makes text contents like text tables, text fields, drawing shapes,.... available. Remember that this service manager is not equal to the service manager which creates the initial connection to OpenOffice.org (cp. *3.2.1 Service Manager*). Each document model has its own service manager.

- text content suppliers

  They offer some features for the text objects.

- objects for styling and numbering

  These services are responsible for the document wide styling and structuring of the text.

Beside these architectural areas there are also overall document aspects of the text document model because it's an office component (cp. *3.2.4 Services*). So a text document model provides also the interfaces `XPrintable`, `XStoreable`, `XReplaceable`, `XRefreshable`, `XModifiable` and `xDocumentInfoSupplier`.

The text document model also prepares access to the graphical user interface and has knowledge about the current view status.
[Open03]

5.1.1 Example 01 – HelloWorld

The first example is the classical „Hello World“[22]. It shows how to write a simple text in an empty swriter – document.

The first step of the ObjectRexx program is to import the required `UNO.cls`. Every example need this class because it offers a function to connect easy to OpenOffice.org, creates the `XMultiComponentFactory`, connects to `BSF.cls` to get the hole BSF4Rexx support, imports all OpenOffice.org interfaces and has some other useful features, e.g. it

[22] „Hello World“ is the name for the simplest possible code in a programming language to print out the string „Hello World“

makes the handling of arrays and the insertig of text to scalc – cells easy. Object Rexx always executes all requires directives before the other sourcecode is executed.

So after importing `UNO.cls` the `Desktop` interface and the `XComponentLoader` has to be created. If there are no parameters passed to `createDesktop()` the default context will be used. In later examples we sometimes need a special context.

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

The `com.sun.star.frame.Desktop` service is the main service for all OpenOffice.org applications. All applications in OpenOffice.org are organized in a hierarchy of frames and the desktop is the root frame for this hierarchy. The interface for the Desktop is the `com.sun.star.frame.XDesktop` and it's provided by the `Desktop`.

To load a component we need the `com.sun.star.frame.XComponentLoader` interface which we get from the `XDesktop`. The `XComponentLoader` offers one single method `loadComponentFromURL()` to load components and this method needs 4 parameters:

- URL

  The URL describes which resource should be loaded. It could be a `file:` URL, a `http:` URL, a `ftp:` URL or a `private:` URL. As you can see, it could be an existing file on your local machine, an existing file anywhere accessible through the network or a new file.

  The URLs for creating new documents are:
  - `"private:factory/swriter"` for an empty swriter component
  - `"private:factory/scalc"` for an empty scalc component
  - `"private:factory/sdraw"` for an empty sdraw component
  - `"private:factory/simpress"` for an empty simpress component

- Target Frame Name

  The target frame name is a reserved name which denotes the frequently used frames in the frame hierarchy. The names are `_blank` for a new frame, `_self` for the frame itself, `_top` for the top frame and so on.

- Search Flag

  The search flag specifies where the frame should be searched.

- Property Values

  The property values indicate how to load the component (e.g. if the component should be loaded „hidden“). You can also load the component without properties. On calling `.UNO~noProps` we get an empty array without any property values.

Now we get the `XTextDocument` interface from the `XComponent`. To get the `XText` interface the `XTextDocument` provides a method called `getText()`. When calling the method `setString("string")` from the interface `XText` we can insert a simple string to an empty swriter document.

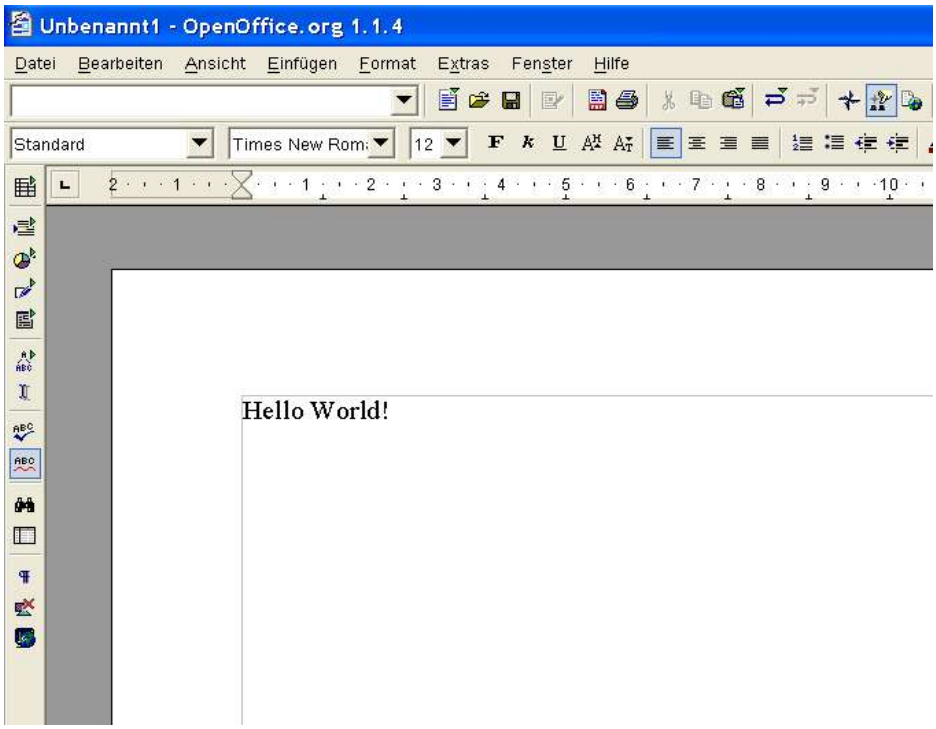

Figure 07: Example 01 – Hello World, Andreas Ahammer

```
-- Example 01
-- open an empty swriter-file and write "Hello World!"

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the OOo Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                          .UNO~noProps)
/* create the TextObject */
xWriterDocument = xWriterComponent~XTextDocument
xText = xWriterDocument~getText()

/* insert text and edit it */
xText~setString("Hello World!")

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

5.1.2 Example 02 – Open an existing swriter-file

The second example is very similar to the first one, but it's loading an existing document. The file we want to open is stored in the same directory like the *.rex – file and is named `textdocument01.sxw` (the extension sxw indicates that the file is an swriter document from OpenOffice.org). So the URL for the method `loadComponentFromURL()` has to be established. The routine `makeUrl` supplies the current directory operating system independent.

```
::routine makeUrl
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

So the correct URL to the file can be received easily.

```
url = makeUrl("textdocument01.sxw")
```

The text document should be loaded „readonly“ (no one can do modifications). Therefore we have to define the properties to load which exists (like many other properties) of a pair of name / value variables.

To create such properties we have to make an array of `PropertyValues` and insert variables called `name` and `value`. BSF4Rexx provides a method to create the arrays:

```
props = bsf.createArray(.UNO~PropertyValue, 1)
```

So the variable `props` is defined as a property value array (with the array length 1). Now we can create an instance of `PropertyValue`, save it in an array, define the elements and insert the variables:

```
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "ReadOnly"
props[1]~Value = box("boolean", .true)
```

The name of the property value is „ReadOnly“ and the value should be „true“. The `box()` method is a type wrapper class to box primitive datatypes into their Java classes counterparts.

```
-- Example 02
-- open an existing swriter-file "readonly"

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the *.sxw - file */
url = makeUrl("textdocument01.sxw")  -- get the document from the current folder

props = bsf.createArray(.UNO~PropertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "ReadOnly"
props[1]~Value = box("boolean", .true)

xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

::requires UNO.cls    -- get UNO support

-- A function for getting the file in the current folder

::routine makeUrl    -- operating system independent
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

5.1.3 Example 03a – Print a page

Example 03a opens an existing swrtier – file and print it. To print a file we need the `XPrintable` interface which every `XComponent` must provide, because an `XComponent` is an `OfficeDocument`.

With this interface it's possible to set the printer with the method `setPrinter()` (if you don't set a printer OpenOffice.org will use your default printer) and to print the document with the method `print()`.

With the `setPrinter()` method you can set property values like the name of the printer, the paper format, the paper size,...

The `print()` method needs some property values where you can specify the number of copies to print, the pages to print, the name of a file to print,...

```
-- Example 03a
-- a simple print example

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the swriter - file  */
url = makeUrl("HelloWorld.sxw")  -- get the document from the current folder
props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~propertyValue~new
props[1]~Name  = "Hidden"
props[1]~Value = box("boolean", .true)
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

/* set the printer */
xPrintable = xWriterComponent~XPrintable

props[1]~Name  = "Name"
props[1]~Value = "hp deskjet 5600 series" -- the name of your printer

xPrintable~setPrinter(props)

/* set the print-options */
props[1]~Name  = "Pages"
props[1]~Value = "1"

/* print current file */
xPrintable~print(props)

::requires UNO.cls    -- get UNO support

-- A function for getting the file in the current folder

::routine makeUrl    -- operating system independent
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

5.1.4 Example 03b – Print a locale html – file

This example is very similar to „Example 03a“, but here we are going to print a html – file.

```
-- Example 03b
-- a simple print example

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface

-- load the HTML file and get its printing interface
```

```
url = makeUrl("aha.html")
xPrintable = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps) -
                            ~xPrintable

xPrintable~print(.UNO~noProps)   -- print the HTML file to default printer

::requires UNO.cls    -- get UNO support

-- a routine for getting the file in the current folder/directory

::routine makeUrl
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

5.1.5 Example 03c – Print a html – file from the internet

This printer – example shows you how to print a page from the internet. Here the first page from http://www.RexxLA.org will be printed.

```
-- Example 03c
-- a simple print example

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface

-- load the document, get printing interface of it
url = "http://www.RexxLA.org" -- now use a URL from the WWW
xPrintable = xComponentLoader~loadComponentFromURL(url, "_blank", 0, .UNO~noProps) -
                            ~xPrintable

/* set a specific printer to print to */
props = bsf.createArray(.UNO~PropertyValue, 1)
props[1] = .UNO~propertyValue~new
props[1]~Name  = "Name"
-- the name of your printer (if not found default one is used)
props[1]~Value = "hp deskjet 5600 series"
xPrintable~setPrinter(props)  -- set the printer

/* set the print-options */
props[1]~Name  = "Pages"
props[1]~Value = "1"       -- just print the first page
xPrintable~print(props)    -- now print the page

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

5.1.6 Example 04 – Statistics 1

This example counts the paragraphs, sentences and words of an existing swriter file. At first, the file is opened hidden from the internet. As you can see the URL of the method `laodComponentsFromURL()` could also be a link to an internet resource:

```
url = "http://www.wu-wien.ac.at/usr/h02c/h0251406/e-commerce/vk6/textdocument01.sxw"
```

The statistics are counted „manually“ with the help of the `XTextCursor`. Manually means, that there are also overall document properties doing this automatically (cp. *5.1.7 Example 05 – Statistics 2*).

The `XText` interface has a method `createTextCursor()` to create a `TextCursor`.

> *„The text cursor travels through the text as a collapsed text range with identical start and end as a point in text, or it can expand while it moves to contain a target string.“ [Open03]*

The `TextCursor` offers some methods to move the cursor, e.g. `goLeft()`, `goRight()`, `gotoStart()`, `gotoEnd()`,...

In the swriter document the `XTextCursor` has three interfaces: `XWordCursor` (a text cursor for words), `XSecntenceCursor` (a text cursor for sentences) and `XParagraphCursor` (a text cursor for paragraphs).

To count paragraphs, sentences and words, the `XTextCursor` is set to „start“ and then the appropriate cursor is going in a loop to the next paragraph, sentence or word and count the passes.

```
-- Example 04
-- use the TextCursor to count the paragrahs, sentences and words of an existing
-- swriter-file

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the sxw-file hidden from the internet */
url = "http://www.wu-wien.ac.at/usr/h02c/h0251406/e-commerce/vk6/textdocument01.sxw"

props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "Hidden"
props[1]~Value = box("boolean", .true)
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

/* create a TextCursor */
xTextCursor = xWriterComponent~XTextDocument~getText~createTextCursor

/* create a Word-, Sentence- and ParagraphCursor */
xWordCursor      = xTextCursor~XWordCursor
xSentenceCursor  = xTextCursor~XSentenceCursor
xParagraphCursor = xTextCursor~XParagraphCursor

/* count paragraphs */
xTextCursor~gotoStart(.false)
paragraphs = 1                -- you start with the first paragraph
DO WHILE xParagraphCursor~gotoNextParagraph(.false) = .true
    paragraphs = paragraphs + 1
END

/* count sentences */
xTextCursor~gotoStart(.false)
sentences = 1                 -- you start with the first sentence
DO WHILE xSentenceCursor~gotoNextSentence(.false) = .true
    sentences = sentences + 1
END

/* count words */
xTextCursor~gotoStart(.false)
words = 1                     -- you start with the first word
DO WHILE xWordCursor~gotoNextWord(.false) = .true
    words = words + 1
END

/* show statistics */
SAY "paragraphs:" paragraphs
SAY "sentences:" sentences
SAY "words:" words
```

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

5.1.7 Example 05 – Statistics 2

This example is going to count the paragraphs, words and characters via the properties of the general document information.

At first we need the interface `XPropertySet` from the `XTextDocument`. Now we are able to set the properties with `setPropertyValue()` and get properties with `getPropertyValue()`. So the instruction

```
xWordCount = xTextDocumentProps~getPropertyValue("WordCount")
```

creates a variable `xWordCount` with the current number of words.

```
-- Example 05
-- get the statistics of a swirter-file from the DocumentProperties

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the *.doc  */
url = makeUrl("textdocument01.sxw")  -- get the document from the current folder
props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "Hidden"
props[1]~Value = box("boolean", .true)
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

/* create the TextObject and get the Properties*/
xTextDocumentProps = xWriterComponent~XTextDocument~XPropertySet

SAY "paragraphs: " xTextDocumentProps~getPropertyValue("ParagraphCount")
SAY "words:      " xTextDocumentProps~getPropertyValue("WordCount")
SAY "characters: " xTextDocumentProps~getPropertyValue("CharacterCount")

::requires UNO.cls    -- get UNO support

-- A function for getting the file in the current folder

::routine makeUrl
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

If you compare the results of „Example 04“ and „Example 05“ you will see, although both examples use the same document they provide different counts. The reason for this is, that in „Example 04“ the cursor is going to count the control characters as normal characters. So e.g. the `WordCursor` take the `ParagraphBreaks` as word.

5.1.8 Example 06 – Text

To insert text in an swriter document and format it, we need again the cursors. If you want to change the color of a the text and insert the text afterwards you'll need the properties of the `XTextCursor`:

```
xTextCursorProps = xTextCursor~XPropertySet
```

Then set the text color:

```
xTextCursorProps~setPropertyValue("CharColor", box("int", "ff 00 ff"x ~c2d))
```

With `"ff 00 ff"x ~c2d` ObjectRexx converts a variable in hexadecimal to integer. This is necessary because `"CharColor"` needs an integer, but colors are easier to handle in hex form. The hex – model uses the RGB code to specify a color. E.g. white = (hex) FFFFFF = (dec)16777215.

If you want to insert the current page number, you'll need the `XPageCursor` because this cursor knows about the page properties. The interface `XModel` from the `XTextDocument` has the method `getCurrentController()` to get the `XController`. The `XController` provides the interface `XTextViewCursorSupplier` with the method `getViewCursor()` to get the `XViewCursor`. From the `XViewCursor` you can get the `XPageCursor` with all its properties.

```
-- Example 06
-- edit the text of a swirter-file

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                         .UNO~noProps)
/* create the TextObject */
xWriterDocument = xWriterComponent~XTextDocument
xText = xWriterDocument~getText

/* create a Text- and a WordCursor */
xTextCursor = xText~createTextCursor
xWordCursor = xTextCursor~XWordCursor

/* create the CursorProperty */
xTextCursorProps = xTextCursor~XPropertySet

/* write the sentence */
xText~setString("This is the text!")
CALL syssleep 2

/* insert the word "old" */
xTextCursor~gotoStart(.false)
xWordCursor~~gotoNextWord(.false)~~gotoNextWord(.false)~~gotoNextWord(.false)
xText~insertString(xWordCursor, "old ", .false)
xTextCursor~gotoEnd(.false)
CALL syssleep 2

/* replace the word "old" */
xTextCursor~gotoStart(.false)
xWordCursor~~gotoNextWord(.false)~~gotoNextWord(.false)~~gotoNextWord(.false) -
           ~~gotoNextWord(.true)
xText~insertString(xWordCursor, "new ", .true) -- .true = replace mode
xTextCursor~gotoEnd(.false)
CALL syssleep 2

/* create a PageCursor */
xPageCursor = xWriterDocument~XModel~getCurrentController ~XTextViewCursorSupplier -
                           ~getViewCursor~XPropertySet~XPageCursor

/* insert the current PageNumber */
xText~insertString(xWordCursor, (" [on page " || xPageCursor~getPage || "]") , -
                  .false)
```

```
/* change the color of the characters */
xTextCursorProps~setPropertyValue("CharColor", box("int", "ff 00 00"x ~c2d))

/* insert a colored smilie */
CALL syssleep 1;   xText~insertString(xWordCursor, " ;", .false)
CALL syssleep 0.5; xText~insertString(xWordCursor, "-", .false)
CALL syssleep 0.5; xText~insertString(xWordCursor, ")", .false)

/* set "modified" false -> UNO will not ask if you want to save the current document
   on close */
xWriterComponent~XModifiable~setModified(.false)
::requires UNO.cls    -- get UNO support
```

5.1.9 Example 07 – Text Table

Text tables in OpenOffice.org consist of rows, rows consist of one or more cells and cells can contain text or rows. The rows are addressed with numbers (1, 2,...) and the columns are addressed with characters (A, B,...). It's also possible to split or merge cells, but then the addressing of the cells is going to be very complex and so you should be careful about this.

To insert a text table into an swriter document the `XMultiServiceFactory` of the `XTextDocument` is needed. The `XMultiServiceFactory` can create the `XTextTable` interface and the method `initialize()` of the `XTextTable` initializes the text table and it appears on the swriter document.

To insert text in the cells with a special format you'll find a routine called `setCellText` that makes the inserting very easy. Such an insert could be:

```
CALL setCellText "A1", "One", xTextTable
```

Another way to insert simple text is to call the cells of the text table with `getCellsByName()` and insert text with `setValue()`.

```
xTextTable~getCellByName("D3")~setValue(6)
```

It's also possible to insert formulas into the cells:

```
xTextTable~getCellByName("A4")~setFormula("sum <A2:A3>")
```

To change the properties of a cell the `XTextTableCursor` is required. It can be created with the method `createCursorByCellName()` of the `XTextTable` and then the properties like BackColor, BottomMargin, Width,.... can be set.

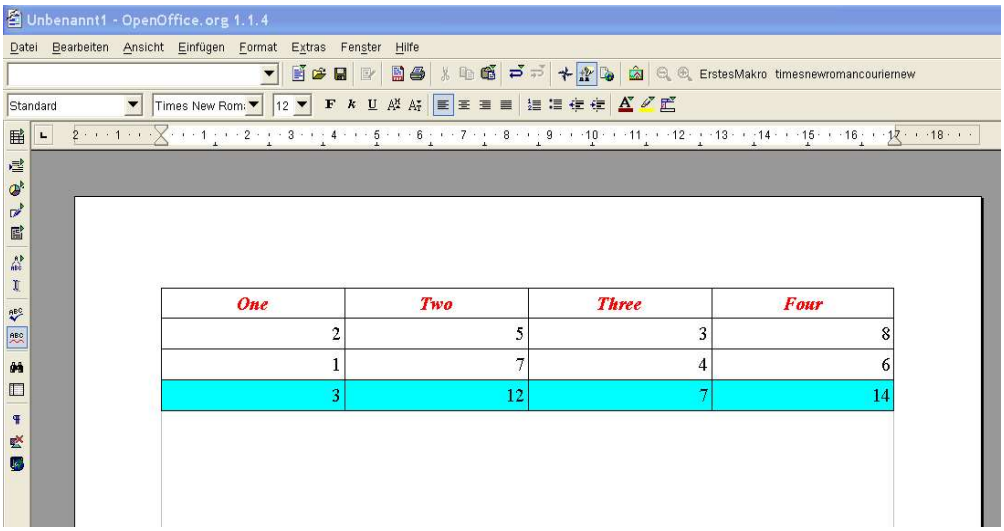

Figure 08: Example 07 – Text Table, Andreas Ahammer

```
-- Example 07
-- insert a table in an empty swriter-file

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                         .UNO~noProps)

/* create the TextObject and the TextCursor */
xTextDocument = xWriterComponent~XTextDocument
xText = XTextDocument~getText
xTextCursor = xText~createTextCursor

/* create the MulitServiceFactory from the current document */
/* (otherwise the created objects cannot be inserted into the document) */
xDMsf = xTextDocument~XMultiServiceFactory

/* create the TextTable */
xTextTable = xDMsf~createInstance("com.sun.star.text.TextTable")~XTextTable
xTextTable~initialize(4, 4)

/* insert TextTable in the Text */
xText~insertTextContent(xTextCursor, xTextTable, .false)

/* insert text in the TextTable */
CALL setCellText "A1", "One",   xTextTable
CALL setCellText "B1", "Two",   xTextTable
CALL setCellText "C1", "Three", xTextTable
CALL setCellText "D1", "Four",  xTextTable

/* insert values in the TextTable */
xTextTable~getCellByName("A2")~setValue(2)
xTextTable~getCellByName("A3")~setValue(1)
xTextTable~getCellByName("B2")~setValue(5)
xTextTable~getCellByName("B3")~setValue(7)
xTextTable~getCellByName("C2")~setValue(3)
xTextTable~getCellByName("C3")~setValue(4)
xTextTable~getCellByName("D2")~setValue(8)
xTextTable~getCellByName("D3")~setValue(6)

/* insert formulars in the TextTable */
xTextTable~getCellByName("A4")~setFormula("sum <A2:A3>")
xTextTable~getCellByName("B4")~setFormula("sum <B2:B3>")
xTextTable~getCellByName("C4")~setFormula("sum <C2:C3>")
xTextTable~getCellByName("D4")~setFormula("sum <D2:D3>")
```

```
/* create the TextTableCursor */
xTextTableCursor = xTextTable~createCursorByCellName("A4")
xTextTableCursorProps = xTextTableCursor~XPropertySet
bgColor = box("int", "00 ff ff"x ~c2d)   -- determine color
xTextTableCursorProps~setPropertyValue("BackColor", bgColor)
xTextTableCursor~gotoCellByName("B4", .true)
xTextTableCursorProps~setPropertyValue("BackColor", bgColor)
xTextTableCursor~gotoCellByName("C4", .true)
xTextTableCursorProps~setPropertyValue("BackColor", bgColor)
xTextTableCursor~gotoCellByName("D4", .true)
xTextTableCursorProps~setPropertyValue("BackColor", bgColor)

::requires UNO.cls    -- get UNO support

/* routine to set the text in a cell
   - cell: the cell you want to write in
   - text: the text you want to write in the cell
   - xTextTable: the table you want to use
*/
::routine setCellText

    use arg cell, text, xTextTable

    xCellText = xTextTable~getCellByName(cell)~XText
    xCellCursor = xCellText~createTextCursor()
    cursorProps = xCellCursor~XPropertySet
    cursorProps~setPropertyValue("CharColor", box("int", "ff 00 00"x ~c2d))
    xCellText~setString(text)
```

5.1.10 Example 08 – Textfield

> *„Text fields are text contents that add a second level of information to text ranges.“ [Open03]*

Such information could be the date, the time, the page count, the file name, the author,...

The first (`xTextFieldTime1`) and the second (`xTextFieldTime2`) text field in the example are the date and the time. Text fields can be created like text tables with the `XMultiServiceFactory` of the current `XComponent`. The difference between `xTextFieldTime1` and `xTextFieldTime2` are the properties. The second text field has the property `IsDate` set to `true`, whereas default the text field `com.sun.star.text.TextField.DateTime` shows the current time.

Between the text fields we insert a control character, more precisely a paragraph break:

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

Therefore the static value `PARAGRAPH_BREAK` from the interface `com.sun.star.text.ControlCharacter` is required. BSF provides the method `getConstant()` to manage this.

The next innovation in this example is the `XModifiable` interface which, similar to the `XPrintable` interface, every `XDocument` has to support. By calling the method `setModified()` you can define with `true` or `false` if OpenOffice.org should ask if you would save the current document on closing.

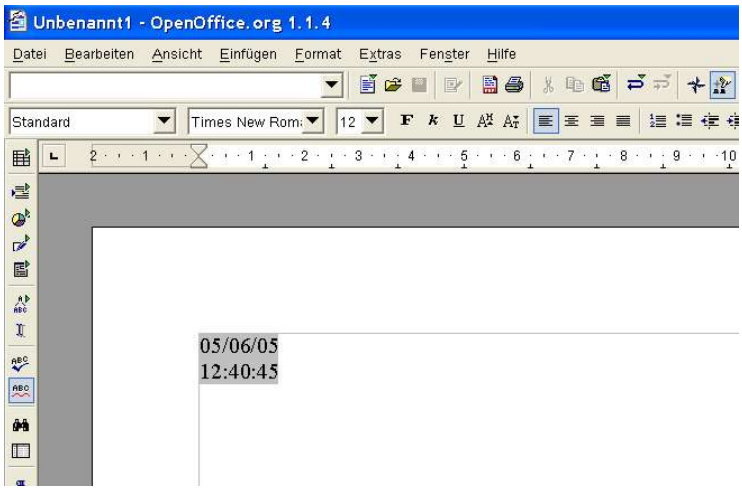

Figure 09: Example 08 – Textfield, Andreas Ahammer

```
-- Example 08
-- insert a textfield (with the current time) in an empty *.sxw - file

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/swriter"
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                          .UNO~noProps)
/* create the TextObject and the TextCursor */
xTextDocument = xWriterComponent~XTextDocument
xText = XTextDocument~getText()
xTextCursor = xText~createTextCursor()

/* create the MulitServiceFactory for the current document */
/* (otherwise the created objects cannot be inserted into the document) */
xDMsf = xTextDocument~XMultiServiceFactory

/* create the TextFields */
xTextFieldTime1 = xDMsf~createInstance("com.sun.star.text.TextField.DateTime") -
                     ~XTextField
-- set the properties for TextFieldTime1- so we can see the date
xTextFieldTime1~XPropertySet~setPropertyValue("IsDate", box("boolean", .true))

xTextFieldTime2 = xDMsf~createInstance("com.sun.star.text.TextField.DateTime") -
                     ~XTextField

/* insert the TextFieldTime1 */
xTextCursor~gotoStart(.false)
xText~insertTextContent(xTextCursor, xTextFieldTime1, .false)

/* insert a "paragraph_break" and the TextFieldTime2 */
xTextCursor~gotoEnd(.false)
xText~insertControlCharacter(xTextCursor, -
      bsf.getConstant("com.sun.star.text.ControlCharacter", "PARAGRAPH_BREAK"), -
      .false)
xText~insertTextContent(xText~getEnd, xTextFieldTime2, .false)

/* set "modified" false -> UNO will not ask to save the current document
   on close */
xWriterComponent~XModifiable~setModified(.false)

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

5.1.11 Example 09 – Search, Replace and Insert Text

Example 09 shows how to search and replace text and how to insert text from another existing document.

At first an existing swriter document is opened and the `XReplaceable` interface is created which provides the method `createReplaceDiscriptor()` to create the `XReplaceDiscriptorInterface`. Now the string to search for, the string to replace and the properties (how to insert the new string, e.g. bold) are defined. At last the replace is done with the current properties.

To insert text from another swriter document the `XDocumentInsertable` interface from the `XTextCursor` is needed. With the method `insertDocumentFromURL()` the insertion can be done.

The last function in this example is to store the current document. The `XStoreable` interface from the `XComponent` has a method `storeAsURL()` which needs an URL and properties to store as a file. The URL comes from the routine `makeUrl` which returns the current folder.

```
-- Example 09
-- you have an inquiry - so make an order...

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = makeUrl("inquiry.sxw")  -- get the document from the current folder
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                         .UNO~noProps)
CALL syssleep 3

/* create the TextObject */
xWriterDocument = xWriterComponent~XTextDocument
xText = xWriterDocument~getText

/* create TextCursors */
xTextCursor = xText~createTextCursor

/* create the ReplaceDescriptor */
xReplaceable = xWriterDocument~xReplaceable
xReplaceDescriptor = xReplaceable~createReplaceDescriptor()

/* search for "Inquiry" and replace it with "Order" in BOLD weight */
xReplaceDescriptor~~setSearchString("inquiry")~~setReplaceString("Order")

props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "CharWeight"
props[1]~Value = box("float", -
                     bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD"))

-- replace text gets formatted in BOLD font weight
xReplaceDescriptor~XPropertyReplace~setReplaceAttributes(props)

-- now do the replace
xReplaceable~replaceAll(xReplaceDescriptor)

/* insert a text from another textdocument */
xTextCursor~gotoEnd(.false)

insertURL = makeUrl("ordertext.sxw") -- get the document from the current folder
xTextCursor~XDocumentInsertable~xDocumentInsertable -
           ~insertDocumentFromURL(insertURL, .UNO~noProps)

/* save the document under a new name */
xWriterComponent~XStorable~storeAsURL(makeUrl("order.sxw"), .UNO~noProps)
```

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

-- A function for getting the file in the current folder

::routine makeUrl
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

5.1.12 Example 10 – Convert a swriter document to pdf

This example is a small and really useful program to convert text documents to pdf.

Here we need the `com.sun.star.lang.XMultiComponentFactory` to create the instance `com.sun.star.ui.dialogs.FilePicker`. So we have to connect to the server and retrieve the `XContext` object and the `XMultiComponentFactory`.

```
xContext = UNO.connect()
XMcf = xContext~getServiceManager
```

Then we can create the `Desktop` within this context.

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

The next step is that the user can declare which existing document should be opened. Therefore we are going to create an instance of the FilePicker of OpenOffice.org

```
fd = xMcf~createInstanceWithContext("com.sun.star.ui.dialogs.FilePicker", xContext)
xfp = fd~XFilePicker
```

and set the properties like the path, the title and the filetypes.

```
urlPath = convertToUrl(directory())             -- convert current directory to URL
xfp~setDisplayDirectory(urlPath)                -- determine directory to start out with
ret = xfp~setTitle(date("S") time()": Please choose a writer file to convert to PDF!")
xfm = xfp~XFilterManager                        -- get the filter manager interface
fileTypes = "*.sxw;*.doc;*.txt;*.text"
xfm~appendFilter("Text files" pp(fileTypes), fileTypes)  -- first filter („current" one)
xfm~appendFilter("All files", "*")                       -- second filter
```

So we get the correct URL to the file which should be converted. The methods `date()` and `time()` return the current date and time. The method `pp()` prints the passed string on the control panel in a nice format.

Then swriter loads the file hidden and stores it with special properties. The important property is the `FilterName` which is set to `writer_pdf_export`. So OpenOffice.org knows which filter should be used to export the file.

You can find a list of all possible import and export filters at [Pito04], page 218.

Notice that with this little program you can convert every file to pdf which can be imported by swriter.

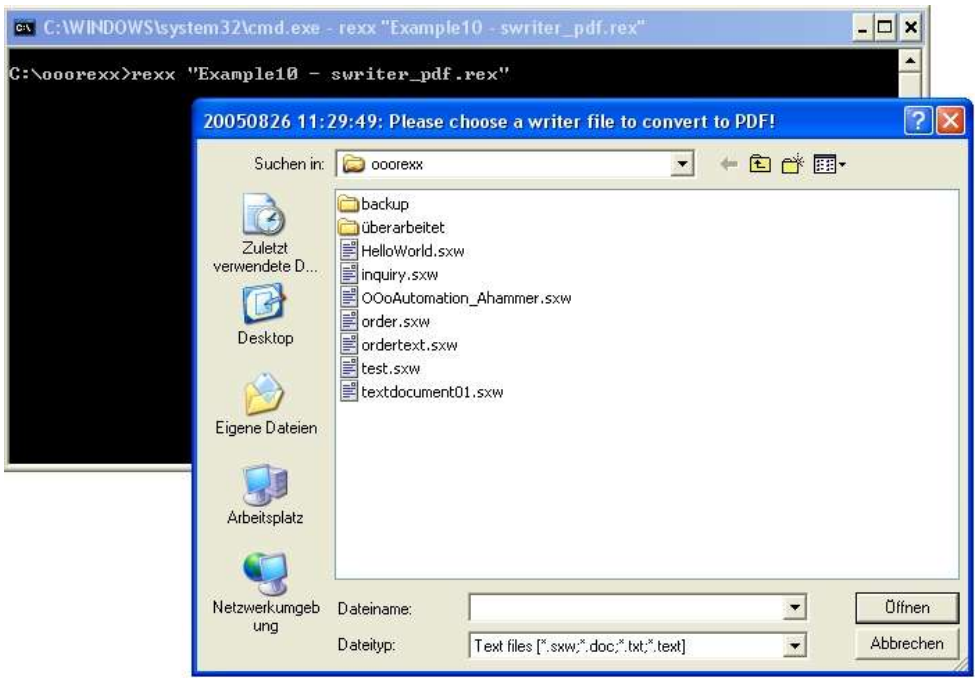

Figure 10: Example 10 – pdf FilePicker, Andreas Ahammer

```
-- Example 10
-- Open an existing textfile and convert it to *.pdf
-- Possible files are *.sxw, *.doc, *.txt,....

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop(xContext) -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* create the FilePickerDialog */
url = getFileName(xMcf, xContext)      -- get file name from user
IF url =.nil THEN EXIT                 -- user did not pick a file
SAY "File:" pp(ConvertFromURL(url)) "..."

props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "Hidden"
props[1]~Value = box("boolean", .true)

xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)
xStorable = xWriterComponent~XStorable -- get xStorable interface

/* convert it to *.pdf */
props = bsf.createArray(.UNO~propertyValue, 2)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "FilterName"
props[1]~Value = "writer_pdf_Export"
props[2] = .UNO~PropertyValue~new
props[2]~Name  = "CompressMode"
props[2]~Value = 2

storeURL = substr(url, 1, lastpos(".", url)) || "pdf"   -- create output file name
xStorable~storeToUrl(storeURL, props)          -- store the file with props to URL
```

```
say "File:" pp(ConvertFromURL(storeURL)) "successfully created!"

::requires UNO.cls    -- get UNO support

::routine getFileName -- FilePicker dialog to ask the user to pick a file to convert
   use arg xMcf, xContext ,oDesktop       -- retrieve multi service factory

   fd = xMcf~-
    createInstanceWithContext("com.sun.star.ui.dialogs.FilePicker", xContext)
   xfp = fd~XFilePicker  -- get the interface for driving the FilePicker
                         -- functionality

   urlPath = convertToUrl(directory())  -- convert current directory to URL
   xfp~setDisplayDirectory(urlPath)     -- determine directory to start out with
   ret = xfp~setTitle(date("S") time() || -
                      ": Please choose a writer file to convert to PDF!")

   xfm = xfp~XFilterManager     -- get the filter manager interface
   fileTypes = "*.sxw;*.doc;*.txt;*.text"
   xfm~appendFilter("Text files" pp(fileTypes), fileTypes) -- first filter is
                                                           -- "current" one
   xfm~appendFilter("All files", "*")

   ret = xfp~execute            -- execute the dialog
   IF ret=1 THEN                -- 1...o.k., 0...cancel
      file = xfp~files[1]

   fd~XComponent~dispose        -- close dialog

   IF ret=1 THEN return file
   return .nil                  -- canceled by user
```

5.2 scalc - examples

Scalc documents consists of spreadsheets, which provide functions to operate on data cells and to do complex calculations. That's why they are called spreadsheet documents.

> *„The core of the spreadsheet document model are the spreadsheets contained in the spreadsheet container. When working with document data, almost everything happens in the spreadsheet objects extracted from the spreadsheets container.“ [Open03]*

The most important interface is the `com.sun.star.sheet.XSpreadsheet` because it provides access to the cells. So we need it for every scalc – example.

5.2.1 Example 11 – Hello World

The first example for scalc application is „Hello World“. It shows how to insert a simple string into the first cell.

The example starts with opening an empty scalc document and get the `XSpreadSheet` interface. Then it is possible to insert a string into a specific cell calling

```
CALL UNO.setCell xSheet, 0, 0, "HelloWorld!"
```

The function `setCell` is predefined in `UNO.cls` and provides an easy way to insert strings and formulas to cells.

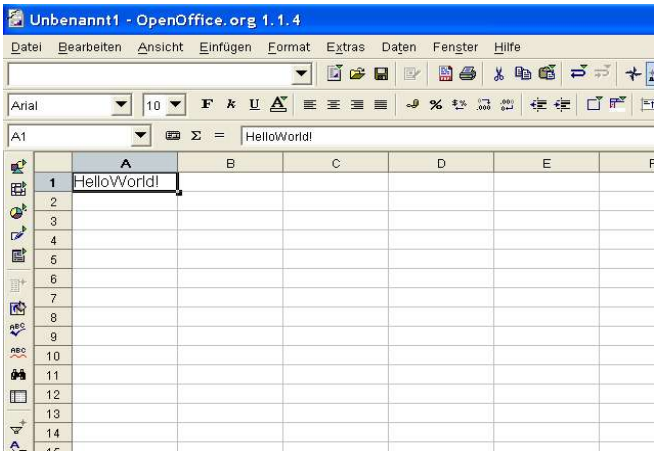

Figure 11: Example 11 – Hello World, Andreas Ahammer

```
-- Example 11
-- open a blank *.sxc - file and insert some text

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                     ~XSpreadSheet

/* insert some text */
CALL UNO.setCell xSheet, 0, 0, "HelloWorld!"

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

5.2.2 Example 12 – Print Area

In Example 12 we want to print a specific area of the spreadsheet document. The printing process is equal to the printing process of the swriter document because both are `OfficeDocuments`.

So we have only to define the area we want to print. At first a `CellRange` is required to specify the area (in the example the area is `"B6:E13"`):

```
myRange = xSheet~XCellRange~getCellRangeByName("B6:E13")
```

Then we have to get the address of the underlying data:

```
myAddr = myRange~XCellRangeAddressable~getRangeAddress
```

and create an array so that we can set the print area of the interface `XPrintAreas`.

In this example we have to load the Java class `"com.sun.star.table.CellRangeAddress"` to make an array, therefore we need the instruction

```
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
```

So we can create a CellRangeAddress – Array like a PropertyValue – Array:

```
oAddr = bsf.createArray(.UNO~CellRangeAddress, 1)
```

Only the cells within the red „border“ are printed:

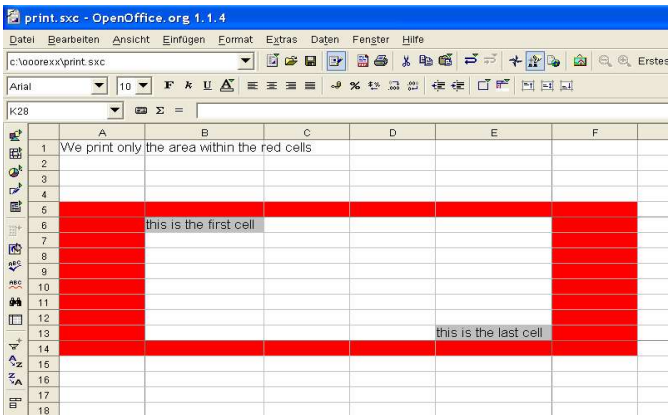

Figure 12: Example 12 – Print Area, Andreas Ahammer

```
-- Example 12
-- open a *.sxw - file and print a specified area 2 times

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface

/* get file from current directory, turn it into an UNO URL */
url = makeUrl("print.sxc")   -- get the document from the current folder
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                     ~XSpreadSheet

/* create a cell range, then get the CellRangeAddress */
myRange = xSheet~XCellRange~getCellRangeByName("B6:E13")
myAddr = myRange~XCellRangeAddressable~getRangeAddress

-- load the required class "com.sun.star.table.CellRangeAddress"
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"

oAddr = bsf.createArray(.UNO~CellRangeAddress, 1) -- create Java array
oAddr[1] = myAddr                                 -- assign CellRangeAddress
xSheet~XPrintAreas~setPrintAreas(oAddr)           -- set PrintAreas

/* set the printer */
xPrintable = xCalcComponent~XPrintable
props = bsf.createArray(.UNO~propertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "Name"
props[1]~Value = "hp deskjet 5600 series" -- the name of your printer
xPrintable~setPrinter(props)
```

```
props[1]~Name  = "CopyCount"
props[1]~Value = box("short", 2)          -- two copies
xPrintable~print(props)                   -- print it

::requires UNO.cls    -- get UNO support

::routine makeUrl    -- operating system independent
  return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

5.2.3 Example 13 – Cell formatting

When formatting cells there are two options to do this:

1. Get the `XCell` with the method `getCellByPostion()` from the `XSpreadSheet` interface and set the properties of a single cell.

2. Make a `XCellRange` from the `XSpreadSheet` interface and set the properties of the `XCellRange`. So all including cells will have the chosen properties.

Note: If you want to insert a formula into a cell, you have to insert the English version of the formula. E.g. to get the current date in German the formula is shown in scalc with `"=Heute()"`, but when automating OpenOffice.org you still have to insert the English version `"=Today()"`.

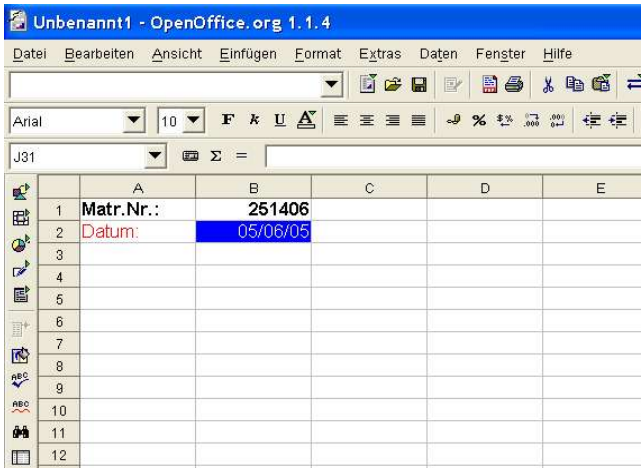

Figure 13: Example 13 – Cell formattings, Andreas Ahammer

```
--- Example 13
-- open a blank *.sxc - file and test the formats

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxc - file */
```

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

/* insert some text */
CALL UNO.setCell xSheet, 0, 0, "Matr.Nr.:"
CALL UNO.setCell xSheet, 1, 0, "0251406"

CALL UNO.setCell xSheet, 0, 1, "Datum:"
-- Remember: even if you have another version of UNO then the english one,
-- you still have to write the formulas in english
CALL UNO.setCell xSheet, 1, 1, "=TODAY()"

/* format the cells via xCell*/
xCell = xSheet~getCellByPosition(1, 1) -- cell "A2"
xPropertySet11 = xCell~XPropertySet~setPropertyValue("CellBackColor", -
                                                     box("int", "00 00 ff"x ~c2d))

xCell = xSheet~getCellByPosition(0, 1)  -- cell "B2"
xPropertySet01 = xCell~XPropertySet~setPropertyValue("CharColor", -
                                                     box("int", "ff 00 00"x ~c2d))

/* format the cells voa xCellRange */
xPropSet = xSheet~XCellRange~getCellRangeByName("A1:B1")~XPropertySet
xPropSet~setPropertyValue("CharWeight", box("float", -
                          bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))

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

5.2.4 Example 14 – Function Autofill

With the method `fillAuto()` from the interface `XCellSeries` it's possible to complete rows and colums automatically. To create the `XCellSeries` interface from a `XCellRange` I wrote a simple routine:

```
::routine getCellSeries
  use arg xSheet, aRange
  return xSheet~XCellRange~getCellRangeByName(aRange)~XCellSeries
```

Now you can call the method `fillAuto()` with the desired properties.

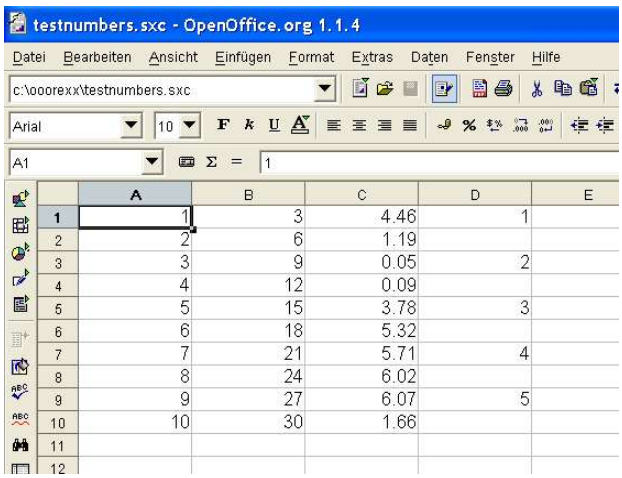

Figure 14: Example 14 – Function Autofill, Andreas Ahammer

```
--- Example 14
-- open a blank *.sxc - file and test the autofill-function

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                     ~XSpreadSheet

/* insert some text */
CALL UNO.setCell xSheet, 0, 0, "1"                 -- cell "A1"
CALL UNO.setCell xSheet, 1, 0, "=(A1*3)"           -- cell "B1"
CALL UNO.setCell xSheet, 2, 0, "=($A$1*10*RAND())" -- cell "C1"
CALL UNO.setCell xSheet, 3, 0, "1"                 -- cell "D1"

/* and AutoFill it */
to_bottom = bsf.getConstant("com.sun.star.sheet.FillDirection", "TO_BOTTOM")
getCellSeries(xSheet, "A1:C10")~fillAuto(to_bottom, 1)
getCellSeries(xSheet, "D1:D10")~fillAuto(to_bottom, 2)

/* save the result - we need it for the next example */
storeURL = makeURL("testnumbers.sxc")  -- save the document in the current folder
xCalcComponent~XStorable~storeAsURL(storeURL, .UNO~noProps)

::requires UNO.cls    -- get UNO support

::routine getCellSeries
  use arg xSheet, aRange
  return xSheet~XCellRange~getCellRangeByName(aRange)~XCellSeries

-- A function for getting the file in the current folder

::routine makeUrl
  return  ConvertToURL(directory() || "\" || arg(1))
```

5.2.5 Example 15 – Sort

A core function of all tables is that they are able to sort in specific ways. The service `com.sun.star.table.TableSortDescriptor` offers such functions to sort a scalc document.

At first an existing document, which was made in „Example 14 – Function Autofill“ is opened and the way to sort the table is defined. In this context we need the `com.sun.star.table.TableSortField` to set the properties and to sort the table by the first column ascending.

Then the properties for the `XSortable` interface are created. Therefore we need the properties of the `TableSortField` and furthermore we set the property `ContainsHeader` to `true`. So the last 9 rows are going to be sorted by the first column ascending.

```
-- Example 15
-- open an existing *.sxc - file, insert a header and sort the first column
```

```
-- ascending

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface

/* open the "testnumbers.sxc" from current folder/directory */
url = ConvertToURL(directory() || "\testnumbers.sxc")
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                      ~XSpreadSheet

/* create the CellRange */
xRange = xSheet~XCellRange~getCellRangeByName("A1:A10")

/* define the fields to sort */
CALL UNO.loadClass "com.sun.star.table.TableSortField" -- load UNO-class

tableSortField = bsf.createArray(.UNO~TableSortField, 1)
tableSortField[1] = .UNO~TableSortField~new
tableSortField[1]~Field           = 1
tableSortField[1]~IsAscending     = .true
tableSortField[1]~IsCaseSensitive = .false

/* define the sort descriptor */
props = bsf.createArray(.UNO~propertyValue, 2)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "ContainsHeader"
props[1]~Value = box("boolean", .true)
props[2] = .UNO~PropertyValue~new
props[2]~Name  = "SortFields"
props[2]~Value = tableSortField

SAY "sleeping, then sorting descendingly first column, leaving first row as" -
    "header alone..."
CALL syssleep 3

/* performe the sorting */
xRange~XSortable~sort(props)    -- perform the sorting

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

5.2.6 Example 16 – Database Function

Scalc documents offer a range of database functions like filter cell ranges with specific properties.

This example inserts a list of names and random numbers into a spreadsheet. The random numbers are made by ObjectRexx because we need numbers in the cells and no formulas - so they are able to filter.

The interface `XFilterDecriptor` can filter rows by defined properties. In our case we want all pairs of name / number where the number is greater than or equal to 2000.

Note that the property `IsNumeric` is still `false` even though we want to filter a number.

At last we set the filter properties, so the method `filter()` knows that we have a header.

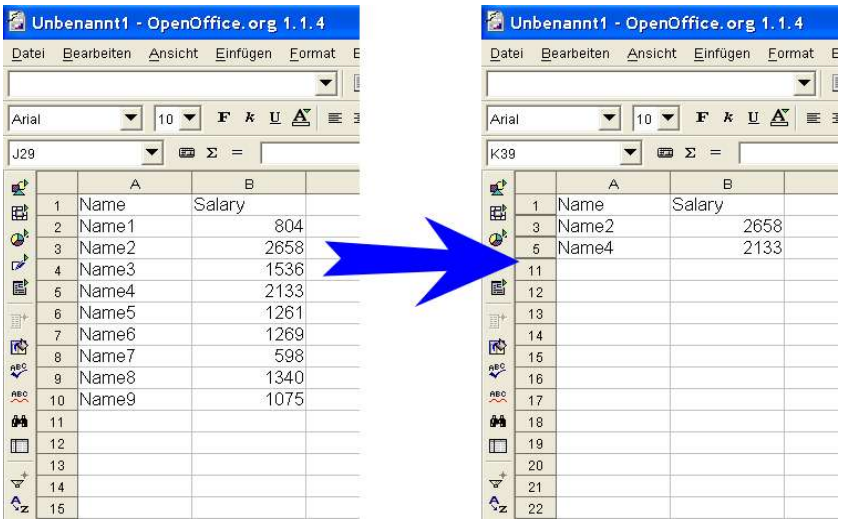

Figure 15: Example 16 – Database Function, Andreas Ahammer

```
-- Example 16
-- open a blank *.sxc, fill it with data and show a database-operation (filtering)

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxc - file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                     ~XSpreadSheet

/* insert some text */
CALL UNO.setCell xSheet, 0, 0, "Name"                      -- cell "A1"
CALL UNO.setCell xSheet, 1, 0, "Salary"                    -- cell "B1"

DO i=1 TO 10
    CALL UNO.setCell xSheet, 0, i, "Name"i             -- cell "A"i
    CALL UNO.setCell xSheet, 1, i, RANDOM(500, 3000)   -- cell "B"i
END

CALL syssleep 3

/* create the CellRange to filter */
xRange      = xSheet~XCellRange~getCellRangeByName("A1:A11")
xFilter     = xRange~XSheetFilterable        -- get filter interface for range
xFilterDesc = xFilter~createFilterDescriptor(.true)

/* define the table filter fields */
CALL UNO.loadClass "com.sun.star.sheet.TableFilterField"
aFilterFields = bsf.createArray(.UNO~TableFilterField, 1)
aFilterFields[1] = .UNO~TableFilterField~new
aFilterFields[1]~Field       = 1
aFilterFields[1]~IsNumeric   = .false
aFilterFields[1]~Operator    = bsf.getConstant("com.sun.star.sheet.FilterOperator",-
                                               "GREATER_EQUAL")
aFilterFields[1]~StringValue = "2000"

xFilterDesc~setFilterFields(aFilterFields)

/* set the FilterProperties and filter */
xFilterProps = xFilterDesc~XPropertySet~setPropertyValue("ContainsHeader", -
                                                         box("boolean", .true))
xFilter~filter(xFilterDesc)
```

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

5.2.7 Example 17 – Document Style

In this example a so called „overall document feature“ is discussed: the style.

> *„A style contains all formatting properties for a specific object. All styles of the same type are contained in a collection named a style family. Each style family has a specific name to identify it in the collection.“ [Open03]*

A new `CellStyle` „MyNewCellStyle“ is created and the style of the cells `A1:D4` is set to this new style.

The `XMultiServiceManager` of the current `XDocument` creates the `XCellStyle` which can be added with the method `insertByName()` to the cell style container. If you want to set cells to the created style, you need the `XPropertySet` interface from a `XCellRange` and call the method `setPropertyValue()`.

```
xCellRange~XPropertySet~setPropertyValue("CellStyle", "MyNewCellStyle")
```

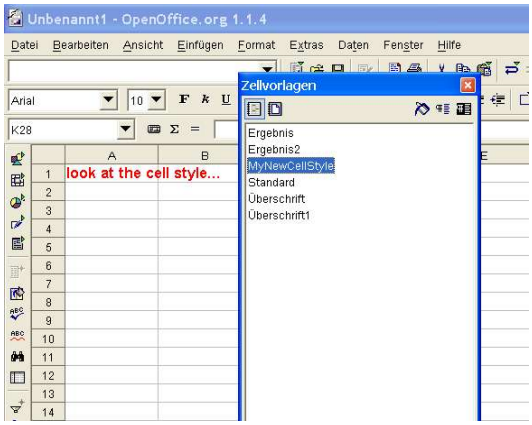

Figure 16: Example 17 – Document Style, Andreas Ahammer

```
-- Example 17
-- open a blank *.sxc and set the CellStyle

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxc - file */
url = "private:factory/scalc"
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                      .UNO~noProps)
/* get the SpreadSheet you want to work with */
xDocument    = xCalcComponent~XSpreadSheetDocument
-- get first sheet in spreadsheet
xSheet = xDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* insert some text */
CALL UNO.setCell xSheet, 0, 0, "look at the cell style..."
```

```
CALL syssleep 2

/* get the cell style container */
xFamiliesSupplier = xDocument~XStyleFamiliesSupplier
xCellStyle = -
  xFamiliesSupplier~getStyleFamilies~getByName("CellStyles")~XNameContainer

/* create a new cell style */
xServiceManager = xDocument~XMultiServiceFactory
oCellStyle = xServiceManager~createInstance("com.sun.star.style.CellStyle")
xCellStyle~insertByName("MyNewCellStyle", oCellStyle)

/* modify properties of the new style */
xPropertySet = oCellStyle~XPropertySet
xPropertySet~setPropertyValue("CharWeight", box("float", -
             bsf.getConstant("com.sun.star.awt.FontWeight", "BOLD")))
xPropertySet~setPropertyValue("CharColor", box("int", "ff 00 00"x ~c2d))
CALL syssleep 2

/* create a CellRange and set the PropertyStyle */
xCellRange = xSheet~getCellRangeByPosition(0, 0, 3, 3)
xCellRange~XPropertySet~setPropertyValue("CellStyle", "MyNewCellStyle")

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

5.2.8 Example 18 – Charts 1

> *„Chart documents produce graphical representations of numeric data. They are always embedded objects inside other OpenOffice.org documents.“ [Open03]*

So it's not mandatory to create charts only in spreadsheet documents. Because spreadsheet documents nearly always hold numeric data and they are common to create charts, the next two examples are about charts.

The `XTableChartsSupplier`, an interface of `XSpreadSheet`, has the method `getCharts()` to access the `XTableCharts`. Adding a new chart to the `XTableCharts` is enabled through the method `addNewByName()`. Among other things this method needs the name of the chart, the frame for chart and the data.

The data is a simple `CellRangeAddress` as already discussed in *5.2.2 Example 12 – Print Area*.

The frame for the chart is the `com.cun.star.awt.Rectangle` service. This service knows the position and the size of the chart. The numeric values are 1/100th mm.

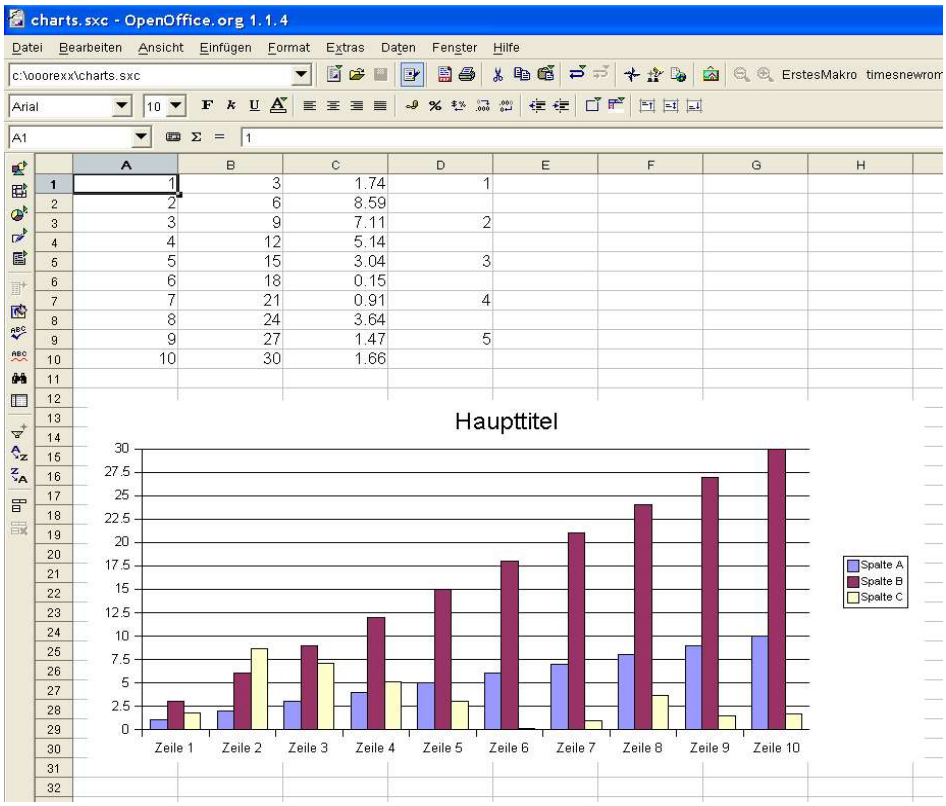

Figure 17: Example 18 – Charts 1, Andreas Ahammer

```
-- Example 18
-- open an existing *.sxc - file and create a simple chart

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the *.sxc - file */
-- get the document from the current folder
url = ConvertToURL(directory()"/testnumbers.sxc")
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
                     ~XSpreadSheet

/* create the frame for the Chart */
oRect = .bsf~new("com.sun.star.awt.Rectangle")
oRect~X      = 300
oRect~Y      = 5000
oRect~Width  = 18000
oRect~Height = 8000

/* catch the underlying data and make a CellRange*/
myRange=xSheet~XCellRange ~getCellRangeByName("A1:C10")
myAddr = myRange~XCellRangeAddressable~getRangeAddress

/* create the CellRangeAddress for the Chart */
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
oAddr = bsf.createArray(.UNO~CellRangeAddress, 1)
oAddr[1] = myAddr

/* get the Sheet's ChartsSupplier, add a new Chart */
xTableCharts = xSheet~XTableChartsSupplier~getCharts
xTableCharts~addNewByName("FirstChart", oRect, oAddr, .true, .true)

/* save the result - we need it for the next example */
storeURL = makeURL("charts.sxc")  -- save the document in the current folder
xCalcComponent~XStorable~storeAsURL(storeURL, .UNO~noProps)
```

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

::routine makeUrl
  return  ConvertToURL(directory() || "\" || arg(1))
```

5.2.9 Example 19 – Charts 2

Here we are going to open the existing file charts.sxc (made by *Example 18 – Charts 1*), get the current chart, make a 3D-Chart and change some properties.

To access an existing chart by name, we need the method `getByName()` from the `XNameAccess` interface, which is provided by the interface `XTableChartsSupplier`. Now we need the interface `XDiagram` to set the properties for the chart. We get it from the method `getDiagram()` of the `XChartDocument` interface, which is provided by the `XComponent`.

At last we set some properties of the interface `X3DDisplay`, an interface used for 3D-Shapes, specially in sdraw.

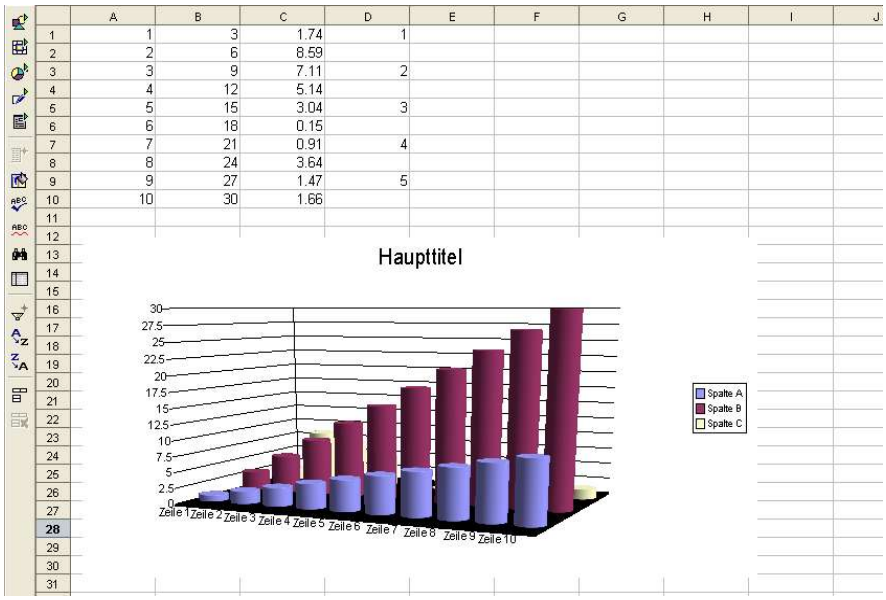

Figure 18: Example 19 – Charts 2, Andreas Ahammer

```
--- Example 19
-- open an existing *.sxc - file, edit a chart and make a 3D-chart

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxc - file */
-- get the document from the current folder
url = ConvertToURL(directory()"/charts.sxc")
xCalcComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* get first sheet in spreadsheet */
xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0) -
```

```
                        ~XSpreadSheet

/* get the Sheet's ChartsSupplier, get first chart */
xTableCharts = xSheet~XTableChartsSupplier~getCharts
xChart = xTableCharts~XNameAccess~getByName("FirstChart")~XTableChart

/* get the embedded Object */
xComponent = xChart~XEmbeddedObjectSupplier~getEmbeddedObject

/* get the Diagram from the ChartDocument */
xDiagram = xComponent~XChartDocument~getDiagram

/* set the properties */
xPropsChart = xDiagram~XPropertySet
CALL syssleep 2
xPropsChart~setPropertyValue("Dim3D", box("bool", .true))
CALL syssleep 2
xPropsChart~setPropertyValue("Deep", box("bool", .true))
CALL syssleep 2
xPropsChart~setPropertyValue("SolidType", box("INT", -
        bsf.getConstant("com.sun.star.chart.ChartSolidType", "CYLINDER")))

/* set the FloorProperties */
xPropsFloor = xDiagram~X3DDisplay~getFloor~XPropertySet
xPropsFloor~setPropertyValue("FillColor", box("int", "00 00 00"x ~c2d))
xPropsFloor~setPropertyValue("FillBackground", box("boolean", .true))

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

5.3 sdraw and simpress – Examples

> *„Draw and Impress are vector-oriented applications with the ability to create drawings and presentations. The drawing capabilities of Draw and Impress are identical. Both programs support a number of different shape types, such as rectangle, text, curve, or graphic shapes, that can be edited and arranged in various ways. Impress offers a presentation functionality where Draw does not. Impress is the ideal application to create and show presentations. It supports special presentation features, such as an enhanced page structure, presentation objects, and many slide transition and object effects. Draw is especially adapted for printed or standalone graphics, whereas Impress is optimized to fit screen dimensions and offers effects for business presentations.“ [Open03]*

5.3.1 Example 20 – Hello World

In the first sdraw example a simple shape, a rectangle, is created which shows the text „HelloWorld“.

A very important interface is the `XDrawPage` because it contains all shapes on a draw page.

With the help of the `XMultiServiceFactory` of the `XComponent` it's possible to create the service `com.sun.star.drawing.RectangleShape` and so the `XShape` interface from the `RectangleShape`.

To insert the `XShape` to the `XDrawPage`, the `XShape` needs a size (a `com.sun.star.awt.Size` service) and a position (a `com.sun.star.awt.Point` service).

Inserting a text into the rectangle is the same procedure like inserting text in a swriter document. The only difference is, that the `XText` interface comes from `XShape` instead of `XDocument`.

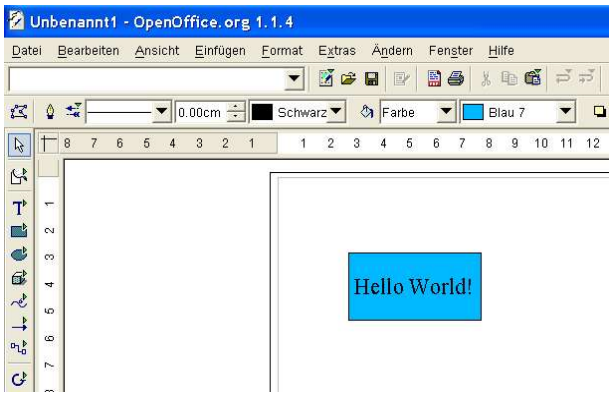

Figure 19: Example 20 – Hello World, Andreas Ahammer

```
-- Example 20
-- open a blank *.sxc and set the CellStyle

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxd - file */
url = "private:factory/sdraw"
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
-- need document's factory to be able to insert created objects
xDocumentFactory = xDrawComponent~XMultiServiceFactory

/* get draw page by index */
xDrawPage = xDrawComponent~XDrawPagesSupplier~getDrawPages~getByIndex(0)~XDrawPage

/* create a Rectangle and add it to the shape */
xShape = xDocumentFactory~createInstance("com.sun.star.drawing.RectangleShape") -
                        ~XShape
-- format the variables
shapeX = 3000
shapeY = 3000
xShape~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))

-- format the variables
shapeWidth  = 5000
shapeHeight = 2500
xShape~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
xDrawPage~add(xShape)  -- add new shape to first draw page

/* now we can insert text into the rectangle */
xShape~XText~setString("Hello World!")

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

5.3.2 Example 21 – Group Function

If you have more than one shape on the `XDrawPage` you can group them to handle all shapes as one object.

At first 2 shapes, a rectangle and an ellipse, are added to the `XDrawPage`. There are a lot of possible properties to set like the corner radius, the shadow, the fill and the line color, the shear and the rotate angle,...

To group the two shapes we need the `XShapes` interface from the `com.sun.star.drawing.ShapeCollection` service. Then we can add both shapes and make a group.

After that we are able to move both object (the group) like one object.

Note that in this example we also need the `XMultiComponentFactory` (c.p. *Example 10 – Convert a swriter document to pdf*).

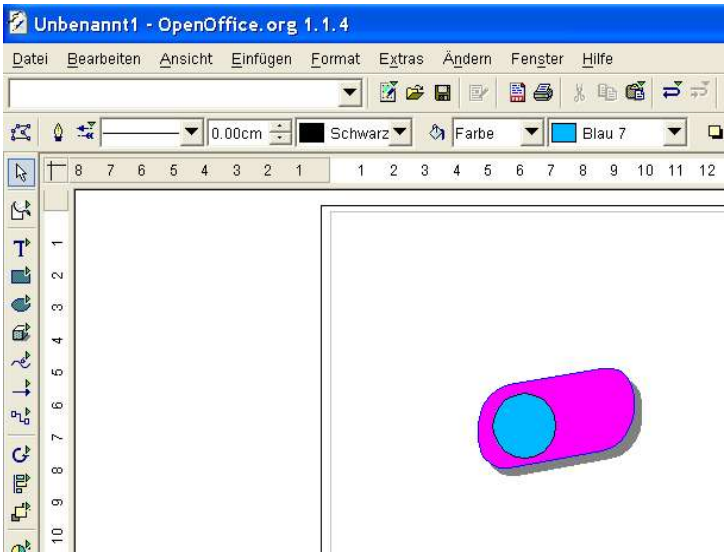

Figure 20: Example 21 – Group Function, Andreas Ahammer

```
-- Example 21
-- handling the properties of a shape and group 2 shapes

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxd - file */
url = "private:factory/sdraw"
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
xDocumentFactory = xDrawComponent~XMultiServiceFactory

/* get draw page by index */
xDrawPage = xDrawComponent~XDrawPagesSupplier~getDrawPages~getByIndex(0)~XDrawPage

/* create a Rectangle and add it to the shape */
shapeRectangle = -
    xDocumentFactory~createInstance("com.sun.star.drawing.RectangleShape")
xShapeRectangle = shapeRectangle~XShape

-- format the variables
shapeX = 3000
shapeY = 3000
```

```
xShapeRectangle~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))

-- format the variables
shapeWidth  = 5000
shapeHeight = 2500
xShapeRectangle~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))

xDrawPage~add(xShapeRectangle)  -- add shape to first draw page

/* set the properties of the rectangle shape */
xShapeProps = xShapeRectangle~XPropertySet

-- set the CornerRadius
xShapeProps~setPropertyValue("CornerRadius", box("int", 1000))

-- set the Shadow
xShapeProps~setPropertyValue("Shadow", box("boolean", .true))
xShapeProps~setPropertyValue("ShadowXDistance", box("int", 250))
xShapeProps~setPropertyValue("ShadowYDistance", box("int", 250))

-- and set the Colors
xShapeProps~setPropertyValue("FillColor", box("int", "ff 00 ff"x ~c2d))
xShapeProps~setPropertyValue("LineColor", box("int", "00 00 ff"x ~c2d))

-- set the Rotation
CALL syssleep 2
xShapeProps~setPropertyValue("ShearAngle", box("int", 1000))
CALL syssleep 2
xShapeProps~setPropertyValue("RotateAngle", box("int", 1000))
CALL syssleep 2

/* insert another shape */
shapeEllipse = xDocumentFactory~createInstance("com.sun.star.drawing.EllipseShape")
xShapeEllipse = shapeEllipse~XShape

shapeX = 3500
shapeY = 3500
xShapeEllipse~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))
shapeWidth  = 2000
shapeHeight = 2000
xShapeEllipse~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))

xDrawPage~add(xShapeEllipse)

/* now group the 2 shapes */
-- get ShapeCollection from the XMultiComponentFactory
xObj = xMcf~-
       createInstanceWithContext("com.sun.star.drawing.ShapeCollection", xContext)
xToGroup = xObj~XShapes
xToGroup~add(xShapeRectangle)
xToGroup~add(xShapeEllipse)
xShapeGrouper = xDrawPage~xShapeGrouper
xShapeGroup = xShapeGrouper~group(xToGroup)

/* so we can move both shapes at once */
CALL syssleep 2
shapeX = 5000
shapeY = 5000
xShapeGroup~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))

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

5.3.3 Example 22 – Fill Function

There are a lot of options to fill shapes and so you can create easy a lot of nice looking objects. The service `com.sun.star.awt.Gradient` offers many properties for the fine tuning of the fill style.

Other properties of the shapes are the distances of the text which can be inserted. E.g. here we insert a text into a rectangle with a left, right, upper and lower distance of precisely 25 mm.

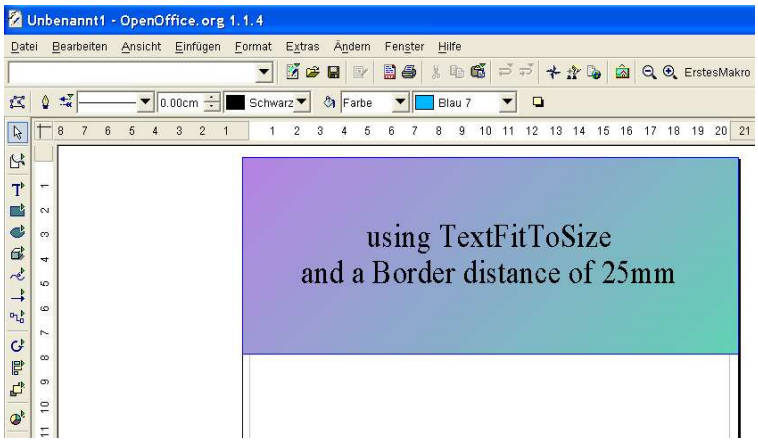

Figure 21: Example 22 – Fill Function, Andreas Ahammer

```
-- Example 22
-- handling the properties of a shape and group 2 shapes

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxd - file */
url = "private:factory/sdraw"
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
xDocumentFactory = xDrawComponent~XMultiServiceFactory

/* get draw page by index */
xDrawPage = xDrawComponent~XDrawPagesSupplier~getDrawPages~getByIndex(0)~XDrawPage

/* create a Rectangle and add it to the shape */
shapeRectangle = -
    xDocumentFactory~createInstance("com.sun.star.drawing.RectangleShape")
xShapeRectangle = shapeRectangle~XShape

-- set the position
shapeX = 0
shapeY = 0
xShapeRectangle~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))

-- set the size
shapeWidth  = 21000
shapeHeight = 8000
xShapeRectangle~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))

-- add the Rectangle to the DrawPage
xDrawPage~add(xShapeRectangle)

/* set the properties of the rectangle shape */
xShapeProps = xShapeRectangle~XPropertySet

-- set the Colors
xShapeProps~setPropertyValue("FillColor", box("int", "ff 00 ff"x ~c2d))
xShapeProps~setPropertyValue("LineColor", box("int", "00 00 ff"x ~c2d))

oGradient = .bsf~new("com.sun.star.awt.Gradient")
oGradient~Style = bsf.getConstant("com.sun.star.awt.GradientStyle", "LINEAR")
oGradient~StartColor     = 95847395
oGradient~EndColor       = 23123123
oGradient~Angle          = 450
oGradient~Border         = 0
oGradient~XOffset        = 0
oGradient~YOffset        = 0
```

```
oGradient~StartIntensity = 100
oGradient~EndIntensity   = 100
oGradient~StepCount      = 10

xShapeProps~setPropertyValue("FillStyle", -
    bsf.getConstant("com.sun.star.drawing.FillStyle", "GRADIENT"))
xShapeProps~setPropertyValue("FillGradient", oGradient)

-- and set the Text
xShapeProps~setPropertyValue("TextFitToSize", -
    bsf.getConstant("com.sun.star.drawing.TextFitToSizeType", "PROPORTIONAL"))
xShapeProps~setPropertyValue("TextLeftDistance",  box("int", 2500))
xShapeProps~setPropertyValue("TextRightDistance", box("int", 2500))
xShapeProps~setPropertyValue("TextUpperDistance", box("int", 2500))
xShapeProps~setPropertyValue("TextLowerDistance", box("int", 2500))

xText = xShapeRectangle~XText
xTextCursor = xText~createTextCursor
xTextCursor~gotoEnd(.false)
xTextRange = xTextCursor~XTextRange
xTextRange~setString("using TextFitToSize")
CALL sysSleep 2

xTextProps = xTextRange~XPropertySet
xTextProps~setPropertyValue("ParaAdjust", -
    bsf.getConstant("com.sun.star.style.ParagraphAdjust", "CENTER"))
xTextProps~setPropertyValue("CharColor", box("int", "ff 00 00"x ~c2d))

xText~insertControlCharacter(xTextCursor, -
    bsf.getConstant("com.sun.star.text.ControlCharacter", "PARAGRAPH_BREAK"), -
    .false)
xTextRange~setString("and a Border distance of 25mm")

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

5.3.4 Example 23 – Convert a sdraw document to jpg

Sdraw documents have like swriter documents a filter function to save the images in another format. In this example an existing sdraw document is opened and all `XDrawPages` are exported to an own *.jpg file.

```
-- Example 23
-- Open an existing drawpage from the internet and export each page to a *.jpg

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the sdraw-file  */
-- get the document from the current folder
url = ConvertToURL(directory()"/drawpage01.sxd")
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                       .UNO~noProps)
/* create and set the filter */
-- get GraphicExportFilter from the XMultiComponentFactory
xGraphicExportFilter = xMcf~-
  createInstanceWithContext("com.sun.star.drawing.GraphicExportFilter", xContext)
xExporter = xGraphicExportFilter~XExporter   -- get the exporter interface
xFilter   = xGraphicExportFilter~XFilter     -- get the filter interface

/* create filter properties and filter it*/
filterprops = bsf.createArray(.UNO~propertyValue, 2)
filterprops[1] = .UNO~PropertyValue~new
filterprops[1]~Name  = "MediaType"
filterprops[1]~Value = "image/jpeg"
filterprops[2] = .UNO~PropertyValue~new
filterprops[2]~Name  = "URL"
```

```
filterprops[2]~Value = storeurl

/* with the XDrawPagesSupplier we can use other page-methods */
drawPages = xDrawComponent~XDrawPagesSupplier~getDrawPages
count = drawPages~getCount()
filename = directory()"\export" -- base filename for JPEGs to be created

DO i=1 to count         -- iterate over pages
    SAY "Converting " i "of" count "page(s)..."

    xComp = drawPages~getByIndex(i-1)~XComponent --get the page by 0-based index
    xExporter~setSourceDocument(xComp)
    -- define filename
    filterprops[2]~Value = ConvertToURL(filename || i || ".jpg")
    xFilter~filter(filterprops)   -- now carry out the conversion ("filtering")
END

xDrawComponent~dispose  -- close the document

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

5.3.5 Example 24 – Start a blank presentation

Simpress documents are generally sdraw documents with a presentation function.

The interface `XPresentationSupplier` has the method `getPresentation()` which provides the `XPresentation` interface. Now the presentation can be started with the default properties.

```
xPresentation~bsf.invoke("start")
```

„start“ is a method in OORexx class „Object“, thus using message `bsf.invoke()` to dispatch „start“ on the Java side.

```
-- Example 24
-- start an empty presentation

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxw - file */
url = "private:factory/simpress"
xImpressComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                          .UNO~noProps)
-- need document's factory to be able to insert created objects
xImpressFactory = xImpressComponent~XMultiServiceFactory

/* start the presentation */
xPresentation = xImpressComponent~XPresentationSupplier~getPresentation
-- "start" is a method in ooRexx class "Object", hence using message
-- "bsf.invoke()" to dispatch "start" on the Java side
xPresentation~bsf.invoke("start")

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

5.3.6 Example 25 – Make a short presentation

When making a presentation it's important to handle the events of the presentation in the right way. When should an object appear? What should happen when someone clicks with the mouse on the object?

If we want the first object appearing directly after starting the presentation, the properties of the draw page has to be set:

```
xDrawPage0Props~setPropertyValue("Change", box("int", 2))
```

In the same way, every object and every slide needs the specific properties. E.g. the properties of the ellipse on the second slide:

```
xEllipseProps = shapeEllipse~XPropertySet
xEllipseProps~setPropertyValue("OnClick", -
       bsf.getConstant("com.sun.star.presentation.ClickAction", "FIRSTPAGE"))
```

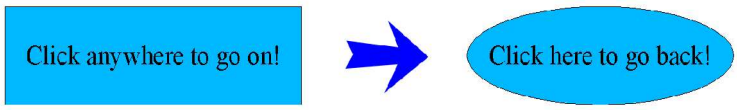

Figure 22: Example 25 – Make a short presentation, Andreas Ahammer

```
-- Example 25
-- make a short presentation

-- Retrieve the Desktop object, we need its XComponentLoader interface to load
-- a new document
oDesktop         = UNO.createDesktop()    -- get the UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- get componentLoader
                                                       -- interface
/* open the blank *.sxi - file */
url = "private:factory/simpress"
xImpressComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, -
                                                          .UNO~noProps)
-- need document's factory to be able to insert created objects
xImpressFactory = xImpressComponent~XMultiServiceFactory

/* get draw page by index */
xDrawPagesSupplier = xImpressComponent~XDrawPagesSupplier
xDrawPages = xDrawPagesSupplier~getDrawPages
xDrawPage0 = xDrawPages~getByIndex(0)~XDrawPage

/* set the properties to animate the first object automatically */
xDrawPage0Props = xDrawPage0~XPropertySet
xDrawPage0Props~setPropertyValue("Change", box("int", 2))
xDrawPages~insertNewByIndex(1)
xDrawPage1 = xDrawPages~getByIndex(1)~XDrawPage

/* create a Rectangle and add it to the DrawPage0 */
shapeRectAngle = -
    xImpressFactory~createInstance("com.sun.star.drawing.RectangleShape")
shapeRectAngle = shapeRectAngle~XShape
shapeX = 3000
shapeY = 3000
shapeRectAngle~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))
shapeWidth  = 10000
shapeHeight = 3000
shapeRectAngle~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
xDrawPage0~add(shapeRectAngle)
shapeRectAngle~XText~setString("Click anywhere to go on!")

/* create a Rectangle and add it to the DrawPage1 */
shapeEllipse = xImpressFactory~createInstance("com.sun.star.drawing.EllipseShape")
xShapeEllipse = shapeEllipse~XShape
```

```
shapeX = 3000
shapeY = 3000
xShapeEllipse~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))
shapeWidth  = 9000
shapeHeight = 3000
xShapeEllipse~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
xDrawPage1~add(xShapeEllipse)
xShapeEllipse~XText~setString("Click here to go back!")

/* set the Animation for the Rectangle on the first page */
xRectangleProps = shapeRectAngle~XPropertySet
xRectangleProps~setPropertyValue("Effect", -
    bsf.getConstant("com.sun.star.presentation.AnimationEffect", -
    "WAVYLINE_FROM_BOTTOM"))

/* set the slide transition effect for the second page */
xSlideProps = xDrawPage1~XPropertySet
xSlideProps~setPropertyValue("Effect", -
    bsf.getConstant("com.sun.star.presentation.FadeEffect", "RANDOM"))
xSlideProps~setPropertyValue("Speed",  -
    bsf.getConstant("com.sun.star.presentation.AnimationSpeed", "MEDIUM"))

/* set the Interaction of the Ellipse on the 2nd Page */
xEllipseProps = shapeEllipse~XPropertySet
xEllipseProps~setPropertyValue("OnClick", -
    bsf.getConstant("com.sun.star.presentation.ClickAction", "FIRSTPAGE"))

/* start the presentation */
xPresentation = xImpressComponent~XPresentationSupplier~getPresentation
-- "start" is a method in ooRexx class "Object", hence using message
-- "bsf.invoke()" to dispatch "start" on the Java side
xPresentation~bsf.invoke("start")

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

5.4 Database

> *„The goal of the OpenOffice.org API database integration is to provide platform independent databases connectivity for OpenOffice.org API. Well it is necessary to access database abstraction layers, such as JDBC and ODBC, it is also desirable to have direct access to arbitrary data sources, if required. … The OpenOffice.org API database integration is based on SQL.“ [Open03]*

The database access in OpenOffice.org 1.1.4 is available through the menu Extras – Data Source. Then the dialog Data Sources Administration opens and you can set all properties of the databases.

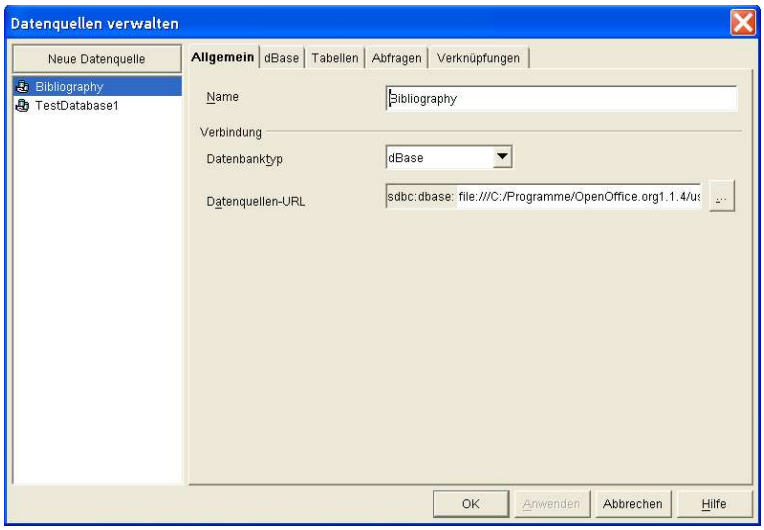

Figure 23: Data Base Administration, Andreas Ahammer

In OpenOffice.org 2.0 the administrations of databases is an own application.

5.4.1 Example 26 – Read data from an existing Database

OpenOffice.org 1.1.4 (and also OpenOffice.org 2.0) has an existing database called „bibliography“. In this example a connection to this database is set up and all data are printed out.

The main procedure is to connect to OpenOffice.org and to the database, create the interface `XRowSet` and set the command type and the command of this interface, execute the command, prepare the `XRow` interface to access the execution and print the data.

```
-- Example 26
-- read data from an existing database

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

-- first we create our RowSet object and get its XRowSet interface
oRowSet = xMcf~createInstanceWithContext("com.sun.star.sdbc.RowSet", xContext)
xRowSet = oRowSet~XRowSet

-- set the properties needed to connect to a database
xProp = xRowSet~XPropertySet

-- the DataSourceName can be a data source registered with [PRODUCTNAME],
-- among other possibilities
xProp~setPropertyValue("DataSourceName", "Bibliography")

-- the CommandType must be TABLE, QUERY or COMMAND - here we use COMMAND
xProp~setPropertyValue("CommandType", -
      box("int", bsf.getStaticValue("com.sun.star.sdb.CommandType", "COMMAND")))

-- the Command could be a table or query name or a SQL command, depending on
-- the CommandType
xProp~setPropertyValue("Command", -
      "SELECT IDENTIFIER, AUTHOR FROM biblio ORDER BY IDENTIFIER")

-- if your database requires logon, you can use the properties User and Password
-- xProp~setPropertyValue("User", "username")
-- xProp~setPropertyValue("Password", "password")

xRowSet~execute
```

```
-- prepare the XRow interface for column access
xRow = oRowSet~XRow

SAY "Identifier | Author"
SAY "-------------------"

DO WHILE xRowSet~next > 0
    ident = xRow~getString(1)
    author = xRow~getString(2)
    say ident || " - " || author
END

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

5.4.2 Example 27 – Handling a MS Access database

Example 27 shows how to get access to an existing MS Access database. At first you have to set up an ODBC bridge to the file `database.mdb` which is included at the examples. How to create an ODBC bridge depends on your operating system. E.g. in the german version of WindowsXP you have to click „Start – Systemsteuerung – Verwaltung – Datenquellen(ODBC)“. Then the following dialog opens:

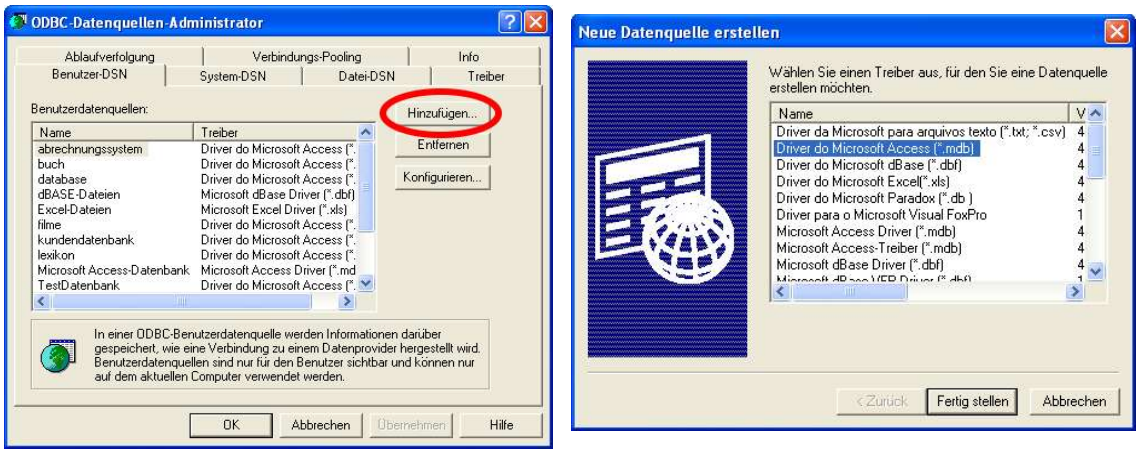

Figure 24: ODBC – Bridge 1, Andreas Ahammer

Choose „Hinzufügen“ and select „Driver do Microsoft Access (*.mdb)“. Then you have to define the name („database“) and the path of the database.

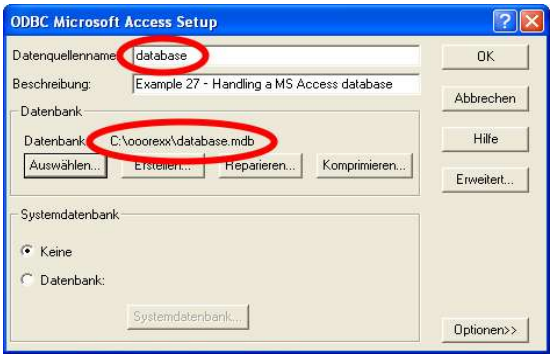

Figure 25: ODBC – Bridge 2, Andreas Ahammer

So the operating system knows where to look if you set up an ODBC – Bridge to „database“.

```
-- Example 27
-- connect to an existing MS-Access database

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

/* get the DriverManager */
oDriverManager = xMcf~-
    createInstanceWithContext("com.sun.star.sdbc.DriverManager", xContext)
xDriverManager = oDriverManager~XDriverManager

/* define the database - URL */
url = "jdbc:odbc:database"

/* set the properties connection */
props = bsf.createArray(.UNO~PropertyValue, 1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "JavaDriverClass"
props[1]~Value = "sun.jdbc.odbc.JdbcOdbcDriver"

/* settle up the connection */
xConnection = xDriverManager~getConnectionWithInfo(url, props)
xStatement = xConnection~createStatement

/* show the content of the database */
CALL printtable xStatement

::requires UNO.cls    -- get UNO support

-- a function to print the content of the database

::routine printtable

  use arg xStatement

  query = "SELECT number, name FROM members"
  xResult = xStatement~executeQuery(query)
  xRow = xResult~XRow

  DO WHILE xResult~next > 0
    number = xRow~getString(1)
    name = xRow~getString(2)
    SAY number || ". " || name
  END
```

5.4.3 Example 28 – Handling a MySQL – Database

To use the following example a MySQL database has to run on your localhost, port 3306 with the appropriate settings (database-name = „test“, user = „root“, password = „“). If you don't have a MySQL database you can easily download and install XAMPP here: http://www.apachefriends.org. Furthermore you have to install the Java – Class `org.gjt.mm.mysql.Driver` which is included in the examples or you can download here http://dev.mysql.com/downloads/connector/j/3.1.html.

Then you have to create the database. Therefore open your browser and connect to http://localhost/phpmyadmin/ (this is the interface for phpmyadmin on your localhost). If the browser can't connect to „localhost“, open the „XAMPP Control Panel“ and start „Apache“ and „MySQL“.

Figure 26: XAMPP Control Panel, Andreas Ahammer

Create a new database with the name „test“:

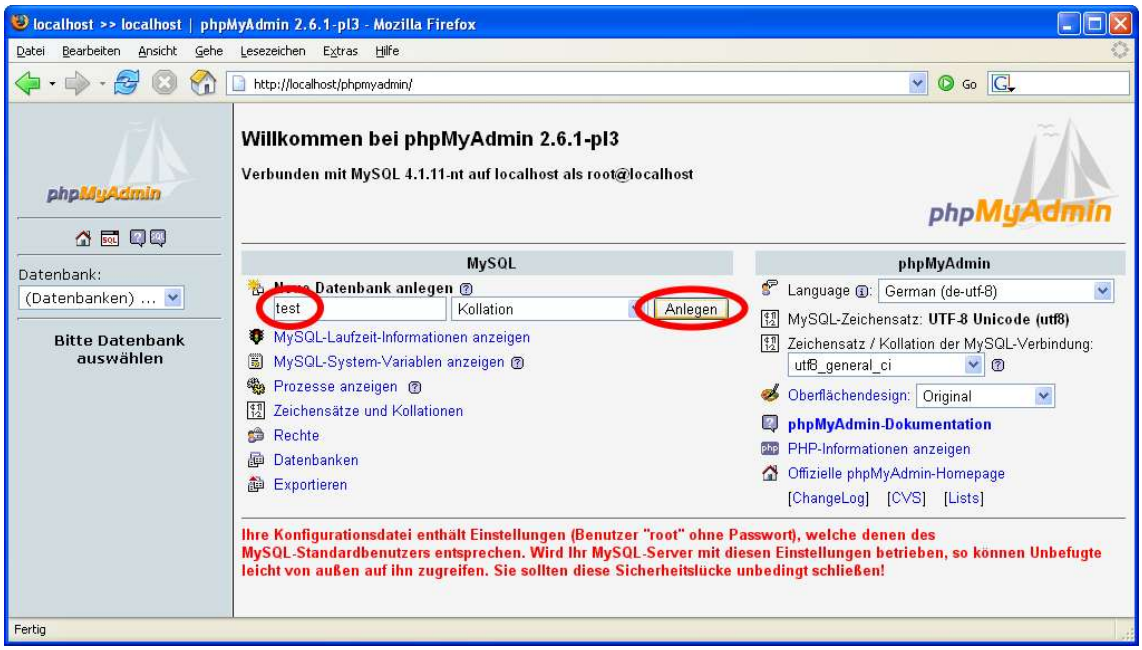

Figure 27: phpMyAdmin, Andreas Ahammer

Now the database is ready to use.

The correct URL to the database is `"jdbc:mysql://localhost:3306/test"`. The properties to set are the username `"root"`, the password `""` and the driver to connect via a JDBC bridge `"org.gjt.mm.mysql.Driver"`. Then the interface `XDriverManager` can create a connection with the method `getConnectionWithInfo()`.

After connecting the interface `XStatement` is created. With the method `executeUpdate()` data can be changed in the database and with the method `executeQuery()` data can be read from the database.

The procedure of the example is to connect to the database, create a table, insert two rows, print the table, insert another row, print the table again and delete the table. To make the code easier the `routine printtable` prints the current table.

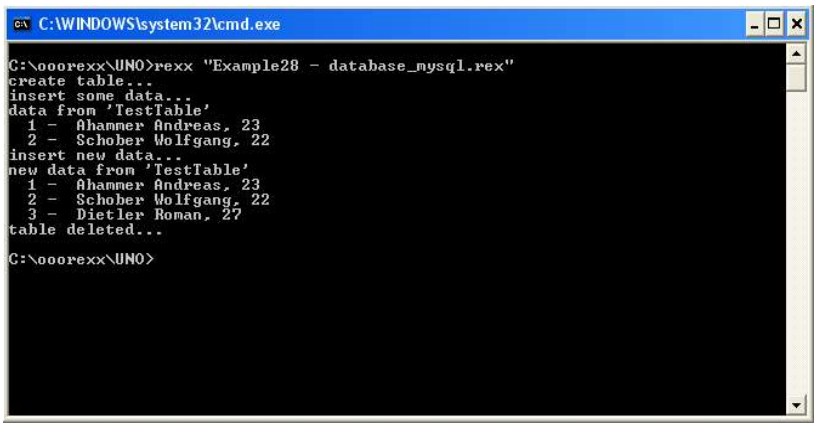

Figure 28: Example 28 – Handling a MySQL – Database, Andreas Ahammer

```
-- Example 28
-- use a MySQL-database

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

/* get the DriverManager */
oDriverManager = xMcf~-
    createInstanceWithContext("com.sun.star.sdbc.DriverManager", xContext)
xDriverManager = oDriverManager~XDriverManager

/* define the database - URL */
url = "jdbc:mysql://localhost:3306/test"

/* set the properties connection */
props = bsf.createArray(.UNO~PropertyValue, 3)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "user"
props[1]~Value = "root"
props[2] = .UNO~PropertyValue~new
props[2]~Name  = "password"
props[2]~Value = ""
props[3] = .UNO~PropertyValue~new
props[3]~Name  = "JavaDriverClass"
props[3]~Value = "org.gjt.mm.mysql.Driver"

/* settle up the connection */
xConnection = xDriverManager~getConnectionWithInfo(url, props)
xStatement = xConnection~createStatement

/* create and show the table */
SAY "create table..."
query = "CREATE TABLE testtable (number INTEGER, firstname VARCHAR(30), " -
        "lastname VARCHAR(30), age INTEGER)"
say "insert some data..."
xStatement~executeUpdate(query)
query = "INSERT INTO testtable VALUES (1, 'Andreas', 'Ahammer', 23)"
xStatement~executeUpdate(query)
query = "INSERT INTO testtable VALUES (2, 'Wolfgang', 'Schober', 22)"
xStatement~executeUpdate(query)

SAY "data from 'TestTable'"
CALL printtable xStatement

SAY "insert new data..."
query = "INSERT INTO testtable VALUES (3, 'Roman', 'Dietler', 27)"
xStatement~executeUpdate(query)

say "new data from 'TestTable'"
call printtable xStatement

/* delete table */
query = "DROP TABLE testtable"
xStatement~executeUpdate(query)
SAY "table deleted..."

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

```
-- A function to show the content of the table

::routine printtable

  use arg xStatement

  query = "SELECT number, firstname, lastname, age FROM testtable"
  xResult = xStatement~executeQuery(query)
  xRow = xResult~XRow

  DO WHILE xResult~next > 0
    number = xRow~getString(1)
    firstname = xRow~getString(2)
    lastname = xRow~getString(3)
    age = xRow~getString(4)
    say "  " || number || " - " lastname || " " || firstname || ", " || age
  END

/*
   NOTE:
   This example will only work if you have a MySQL database
   running on port 3306 named "Test" with an appropriate setup
   and correct tables! Furthermore the class
   "org.gjt.mm.mysql.Driver" has to be installed!
*/
```

5.5 Linguistics

> *„The Linguistic API provides a set of UNO services used for spellchecking, hyphenation or accessing a thesaurus.“ [Open03]*

5.5.1 Example 29 – Dictionary

The last example is a simple dictionary. A user can enter a word and OpenOffice.org checks if this word is valid or not. If it's not valid, a list of alternatives is given.

At first we have to import the Java class `"com.sun.star.lang.Locale"`

```
CALL UNO.loadClass "com.sun.star.lang.Locale"
aLocale = .OOo~Locale~new("en", "US", "")
```

and check if the word is valid:

```
isCorrect = xSpellChecker~isValid(aWord, aLocale, .UNO~noProps)
```

If the word is not correct we can get all alternatives:

```
xSpellAlternatives = xSpellChecker~spell(aWord, aLocale, .UNO~noProps)
```

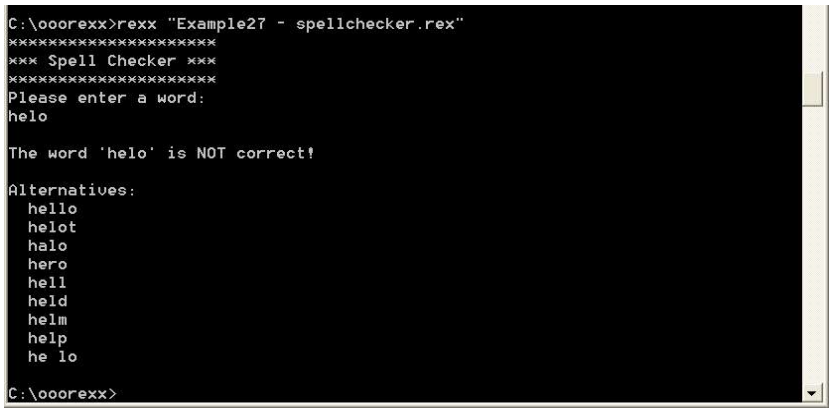

Figure 29: Example 29 – Dictionary, Andreas Ahammer

```
-- Example 29
-- use the SpellChecker to check a word

/* initialize connection to server, get XContext */
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
XMcf = xContext~getServiceManager   -- retrieve XMultiComponentFactory

SAY "**********"
SAY "*** Spell Checker ***"
SAY "**********"

SAY "Please enter a word: "
parse pull aWord              -- get word to spell check
SAY

/* create the LinguServiceManager and the SpellChecker */
mxLinguSvcMgrName = xMcf~-
 createInstanceWithContext("com.sun.star.linguistic2.LinguServiceManager", xContext)
xSpellChecker = mxLinguSvcMgrName~XLinguServiceManager~getSpellChecker

/* load the required class "com.sun.star.lang.Locale" and set the language to
   US-english */
CALL UNO.loadClass "com.sun.star.lang.Locale"
aLocale = .UNO~Locale~new("en", "US", "")

/* test the word if it is valid */
isCorrect = xSpellChecker~isValid(aWord, aLocale, .UNO~noProps)

wordCorrect = ""
IF isCorrect = 0 THEN wordCorrect = "NOT "

say "The word '" || aWord || "' is " || wordCorrect || "correct!"

/* if the word is not correct submit all alternatives */
xSpellAlternatives = xSpellChecker~spell(aWord, aLocale, .UNO~noProps)

IF xSpellAlternatives <> .nil THEN
DO
    SAY
    SAY "Alternatives: "
   DO alternative OVER xSpellAlternatives~getAlternatives
            SAY "  " || alternative
   END
END

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

Note: If the program always returns the result „true“ (even if the word is definitely wrong), check your „Language Settings“ of OpenOffice.org. This can be done from every OpenOffice.org application under „Tools – Options – Language Settings – Languages“. Then choose as language „English“. Now the SpellChecker should work correct.

6 Conclusion

Automating OpenOffice.org via ObjectRexx is definitely a great possibility to make the working with OpenOffice.org more efficient. Because ObjectRexx has access to all Java classes there can be written really useful applications with many functions that without this technology hardly can be developed.

At the moment (in November 2005) there is an ongoing development of all the required techniques. Since October 2005 OpenOffice.org is available in the version 2.0. ObjectRexx was going open source early in the year 2005, so everybody can download it for free and the community could increase. BSF4Rexx is also available in a new version and provides a lot of functions specially for OpenOffice.org automation.

As you can see, all technologies are still alive and I think they will become more important in future.

But there are also a few things which should be enhanced.

At first, it's not easy to install all the required components correct, modify them, set options and classpathes, so that the automation can work well.

Another disadvantage is that all examples above are a „know how, know where“ - story. If you read the literature about ObjectRexx, study the object model, know about the syntax of ObjectRexx and try to create your own examples you will see, that the examples are not really difficult to understand. But they are difficult to develop, because you need to know where to get the specific interface, which are the properties of a service, what's about the class hierarchy and so on. The API of OpenOffice.org is helpful, no question. But I was not always satisfied about it, and so I think, that's the main point which should be improved.

Finally I'll say, OpenOffice.org automation is a helpful and interesting topic and with a little programming experience there could be developed great features.

7 List of references

[Augu05] Augustin, Walter: Examples for Open Office Automation with Scripting Languages (2005), http://wi.wu-wien.ac.at/Studium/LVA-Unterlagen/rgf/autojava/bsf.ooffice/ , retrieved on 2005-11-06

[Apac05] Apache Software Foundation, http://jakarta.apache.org/bsf/, retrieved on 2005-11-06

[at-mix05] at-mix, http://www.at-mix.de/openoffice_punkt_org.htm, retrieved on 2005-11-06

[Chay05] Chay, Julian / Rexx Language Assosiation: http://www.oorexx.org/, retrieved on 2005-11-06

[Flat04] Flatscher, Rony G.: Camouflaging Java as Object REXX, at http://wi.wu-wien.ac.at/rgf/rexx/orx15/2004_orx15_bsf-orx-layer.pdf ; 2004,

[Flat05a] Flatscher, Rony G.: Java Automation - Course slides (in German), at: http://wwwi.wu-wien.ac.at/Studium/LVA-Unterlagen/rgf/autojava/folien/; 2004; retrieved on2005-11-06

[Flat05b] Flatscher, Rony G.: "Automating OpenOffice.org with OORexx: OORexx nutshell exmaples for write and calc", http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_NutShell_OOo.pdf; retrieved on 2005-11-06

[Flat05c] Flatscher, Rony G.: "Automating OpenOffice.org with OORexx: architecture, gluing to rexx using BSF4Rexx", http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf; retrieved on 2005-11-06

[Hahnek05] Hahnekamp, Rainer: Extending the scripting abilities of OpenOffice.org with BSF and JSR-223; course paper, Vienna University of Economics and Business Administration, Information Systems and Operations (Flatscher, Rony G.); January, 2005

[IBM05] IBM: IBM Rexx Family Homepage, at: http://www-306.ibm.com/software/awdtools/rexx; retrieved on 2005-11-06

[Mart05] Martin, Dave J.: Rexx Frequently Asked Questions, http://www.mindspring.com/~dave_martin/RexxFAQ.html, retrieved on 2005-11-06

[Muel01] Müller, Lutz: Einführung in die Programmiersprache Rexx (in German), 2001, http://lptp7.gm.fh-koeln.de/schulung/LUTZ/Rexx12/REXX12.HTM, retrieved on 2005-11-06

[Open03] OpenOffice.org: OpenOffice.org 1.1 - Developer's Guide, http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html, retrieved on 2005-11-06

[Open05] OpenOffice.org: http://de.openoffice.org/product/index.html, retrieved on 2005-11-06

[Pito04] Pitonyak, Andrew: OpenOffice.org Macros Explained (2004)

[RexxLA05] The Rexx Language Association Homepage, at: http://www.rexxla.org; retrieved on 2005-11-06

[Sun05] Sun Microsystems, Inc., http://java.sun.com/, retr.on 2005-11-06

8 Download Links

BSF – Bean Scripting Framework
http://jakarta.apache.org/site/binindex.cgi#bsf

BSF4Rexx – Flatscher, Rony G.: Bean Scripting Framework for Rexx
http://wi.wu-wien.ac.at/rgf/rexx/BSF4Rexx/

J2SE – Sun's Java 2 Standard Edition
http://java.sun.com/j2se/1.5.0/download.html

MySQL – JDBC Driver
http://dev.mysql.com/downloads/connector/j/3.1.html

OORexx – OpenObjectRexx website
http://www.oorexx.com

OOo – OpenOffice.org – Download Central
http://download.openoffice.org

Open Office SDK – OpenOffice.org – Software Development Kit
http://www.openoffice.org/dev_docs/source/sdk/index.html

OOo API – OpenOffice.org application programming interface
http://api.openoffice.org

XAMPP – Apache and MySQL Server
http://www.apachefriends.org

Example scripts from this document and a presentation of this topic:
http://www.wu-wien.ac.at/usr/h02c/h0251406/e-commerce/vk6/

All links have the status of November 2005.

9 Index