Vienna University of Economics and

Business Administration

BACHELOR THESIS

Title:

Facilitate Data Access in OpenOffice.org using ooRexx

Autor: Stefan Schmid

Matriculation Number: 0252354

Field of Study: J033 526 Bakkalaureat Wirtschaftsinformatik

Course: 1236 Vertiefungskurs VI/Bakkalaureatsarbeit - Electronic Commerce

Text Language: English

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

Date: 2007-02-06

I assure

that I have composed this bachelor thesis independently.

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

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

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

_____ _____

Date Signature

Table of Contents

1. Introduction....5
1.1 Abstract....5
1.2 Research Question....5
1.3 Snippet Definition....5

2 Databases....7
2.1 Terms and Definitions....7
2.2 Entity Relationship Diagram....9
2.3 Used Databases....10
2.3.1 MySQL....10
2.3.2 Adabas D....11

3 The technical environment....13
3.1 The Scripting Language Open Object Rexx....13
3.1.1 History....13
3.1.2 Overview....14
3.1.3 Syntax....14
3.2 The Bean Scripting Framework for Rexx....16
3.2.1 BSF....16
3.2.2 BSF4Rexx....17
3.2.2.1 History....17
3.2.2.2 Usage....17
3.3 OpenOffice.org....20
3.3.1 History....20
3.3.2 Overview....21
3.3.3 Architecture....22
3.3.3.1 UNO – The Base Component Technology....22
3.3.3.2 UNO Service Components....23
3.3.3.2.1 Service Manager and Service Objects....23
3.3.3.2.2 Interfaces....24

3.4 Overall Concept....27

4 Database Access....29
4.1 DatabaseContext....29
4.2 DataSources....31
4.2.1 Adding a Adabas D data source to the database context....32
4.2.2 Adding a MySQL data source to the database context....36
4.2.3 Adding other data sources to the database context....38
4.3 Connections....39
4.3.1 Connection through a registered data source....39
4.3.2 Connecting using the DriverManager....42
4.4 Manipulate and Query Data....43
4.4.1 The statement object....44
4.4.2 The RowSet Service....50

5 Additional Database Snippet....54
5.1 Query Definition....54
5.1.1 Store a Query Definition....54
5.1.2 Execute a Query Definition....57

6 Forms....60
6.1 Create a New Form....60
6.1.1 Create a new Writer document....67
6.1.2 Create Components Inside the Master Form....68
6.1.3 Bind Forms to the Database....72
6.1.4 Create Components inside the Sub Form....73
6.1.5 Switch to the Live Mode....75
6.2 Add Form to the Database Document....77

7 Conclusion....80

8 List of Snippets....81

9 List of Illustrations....82

10 Bibliography.....84

1. Introduction

This chapter provides an overview of this paper as well as the research question and a definition of snippets.

1.1 Abstract

The thesis deals with OpenOffice.org automation in view of data access using the scripting programming language Open Object Rexx. This data access includes the communication of OpenOffice.org with external databases, address books and even the creation of forms.

Firstly some theoretically background is given about databases with an focus on relational databases, since they are used in snippets of this thesis. Furthermore some installation instructions are given for the used databases.

The next chapter deals witch all the technical prerequisites that are required to accomplish an OpenOffice.org automation.

After the theoretically requirements have been presented, the following chapters provide several snippets regarding to the access of data sources. They also include examples for automatically creating a predefined query and a form.

1.2 Research Question

How can data sources automatically be accessed inside OpenOffice.org by using Open Object Rexx?

1.3 Snippet Definition

Since this thesis' aim is to provide several snippets, it is important to know what a snippet in this context actually is. In this paper a snippet is defined as a short, runnable program which shows the solution of a specific problem. There is already a

project at http://codesnippets.services.openoffice.org/, which contains various snippets for OpenOffice.org. Snippets of this thesis will also be available there.

2 Databases

Since this thesis is generally about how to establish connections to databases and to work with databases within OpenOffice.org it is essential to have some basic knowledge about databases. This chapter provides at first some terms and definitions concerning databases and is followed by a short explanation about ER-diagrams including the presentation of the used data model. Finally there are also instructions for the installation of the used databases.

2.1 Terms and Definitions

A **database** is a central, managed collection of data, which is made usable by application depended access methods. The **database management system (DBMS)** is a software designed to administrate this collection of data. This includes e.g. the definition of data types, attributes, access rights, etc. Additional, the DBMS enables multiple users and application a concurrent access to the administrated data collection. A **database system** consists of a database, a DBMS and some additional applications, which facilitate editing, managing and analysis of the saved data. OpenOffice.org's Base application can be considered as such a database system. [HaNe05] p.194

A **data model** describes the structure of databases and the way data is stored. There are several data models such as the hierarchical model, the network model and the object database model. Some of them, such as the hierarchical model and the network model, are old and outdated whereas the object database model is a rather new concept inspired by the object oriented paradigm. However the most common data model these days is the relational model and also the databases used in this thesis follow this model.

The **relational model** concept was theoretically invented by the IBM employee E. F. Codd in 1970. However, due to the lack of hardware resources in these days, the first implementation was made not until 1978. The basic concept of the relational model is a two-dimensional table consisting of rows and columns whereas a row is

sometimes also called “tuple”. In this table data is saved as shown in illustration 2.1. It presents a table where data of customers are stored. [Geis05]

cid	firstname	lastname	birthdate
1	Tyler	Durden	02.07.73
2	Jacob	Fuller	03.09.46
3	Mickey	Knox	17.03.76
4	Vincent	Vega	01.12.63

Illustration 2.1: Customer Table.

Relations between tables are established with the use of primary keys and foreign keys. A primary key must be non-ambiguous so that each tuple is unequivocally characterized. In the table shown above the primary key would be the data field *customer id (cid)*. Using this key it is possible to create relations to other tables. [Geis05]

Illustration 2.2 shows how the tables *customer* and *sales* are connected.

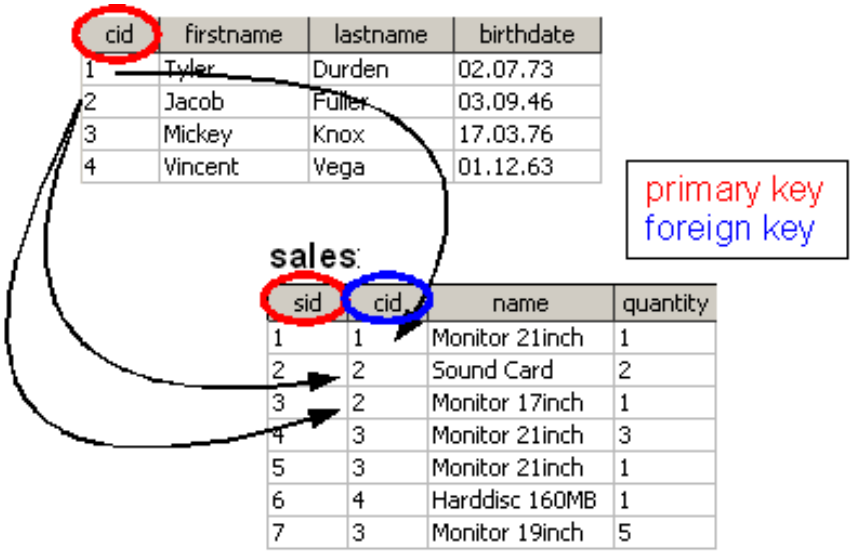

Illustration 2.2: Connection of Tables Customer and Sales.

The primary key of the customer table (cid) is the foreign key of the sales table. Furthermore the sales table has a primary key (sid) to provide an unambiguous ID. As a result a connection between these two tables is created and now it is possible to find out which customer bought which products, stored under which sales ID.

2.2 Entity Relationship Diagram

There are several methods and diagrams for modeling databases. One of the most common notations for the conceptional description of relational databases is the Entity-Relationship-Diagram (ER-Diagram). In this chapter a short introduction of ER-Diagrams is provided, used to describe the model of a computer shop, which is used in several snippets below.

Generally a model is used to represent a slice of reality. This is done by abstraction i.e. to only retain information which is relevant for the modeled problem. Illustration 2.3 shows an ER-Model of a computer shop.

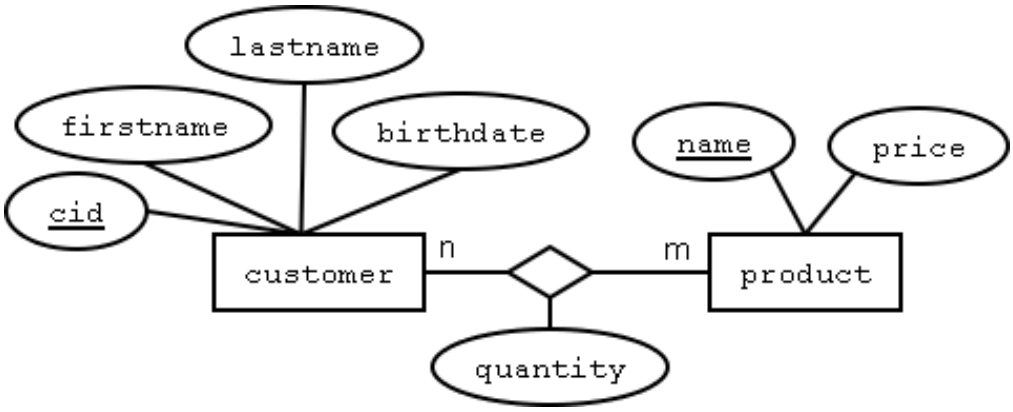

Illustration 2.3: ER-Diagram of a Computer Shop.

Basically an ER-Diagram consists of entities (represented by rectangles), holding attributes (represented by ellipses) and connected via relationships (represented by diamonds). The attribute of an entity which is used as a primary key is underlined like 'cid'. Additionally ER-Diagrams provide the possibility to indicate the number of times each entity participates in relationships. E.g. the customer-product relationship has an cardinality of m:n, which means that one customer can buy several products or one product can be bought by several customers. [SKSu02]

A database that conforms to an ER-Diagram can be represented by a collection of tables. For each entity and for relationships, which have a cardinality of m:n, a table

with an unique table name is created. These tables consist of multiple columns, which are equivalent to the attributes of the associated entity. The conversion of the diagram to the table format is required to implement this model to a database. Illustration 2.4 shows the tables resulting from the transformation.

customer:

cid	firstname	lastname	birthdate
1	Tyler	Durden	02.07.73
2	Jacob	Fuller	03.09.46
3	Mickey	Knox	17.03.76
4	Vincent	Vega	01.12.63

product:

name	price
Monitor 17inch	214
Monitor 19inch	419
Monitor 21inch	759
Harddisc 160MB	67
Sound Card	43
SDRAM 1024MB	156

sales:

sid	cid	name	quantity
1	1	Monitor 21inch	1
2	2	Sound Card	2
3	2	Monitor 17inch	1
4	3	Monitor 21inch	3
5	3	Monitor 21inch	1
6	4	Harddisc 160MB	1
7	3	Monitor 19inch	5

Illustration 2.4: Table Format of the Computer Shop ER-Diagram.

2.3 Used Databases

In this chapter the installation procedure of the databases used in the snippets is described. It is necessary to follow the installation steps in order to get the snippets work, which use the specified databases.

2.3.1 MySQL

The easiest way to get MySQL run on your system is to download and install XAMPP[1], which is a package containing Apache, MySql, PHP and Perl.

You can download XAMMP here: http://www.apachefriends.org/en/xampp.html

[1] XAMPP is provided by Apache Friends a non-profit project to promote the Apache web server.

Afterwards a JDBC driver has to be installed which is needed by OpenOffice.org to establish a connection to MySQL. The driver used in the snippets of this thesis is a Java class called: “com.mysql.jdbc.Driver”. Sometimes in other sources beyond this thesis the driver “org.gjt.mm.mysql.Driver” is used. This driver was an old implementation and its name still exists because of backwards compatibility reasons, but it actually uses the code of the first mentioned driver “com.mysql.jdbc.Driver” now.

A jar package containing this driver can be downloaded here:
http://dev.mysql.com/downloads/connector/j/5.0.html

This jar file has to be registered in OpenOffice.org under *Tools* → *Options* → *OpenOffice.org* → *Java* by pressing the button *“Class Path...”* and afterwards the button *“Add Archive...”*.

In the examples of this thesis the MySQL database “test” is used, which comes with the distribution of XAMPP by default. However, the user “stefan” with the password “apple” is used in all MySQL snippets. This user can be created by going to the URL “localhost” in your web browser. Then you have to choose on the left frame *“phpMyAdmin”* afterwards *“Privileges”* and *“Add a new user”*. Provide the appropriate data there and select all global privileges. Bear in mind that XAMPP has to run during the whole process.

2.3.2 Adabas D

Download the Personal Edition of Adabas D 13. for you operating system and the license key in ZIP or XML format here:
http://www.softwareag.com/Corporate/products/adabas/adad/download/default.asp

Execute Setup.exe, choose Standard setup and give the required information about the XML license file. Next create an environment variable named DBROOT and set as value the path to your installation folder (e.g. C:\Adabas D 13.01). Afterwards restart OpenOffice.org (also close the Quickstarter).

The distribution of Adabas D already comes with a sample database called “**mydb**” which is used in this thesis. This database needs the following data to be accessed:

User: demo

Password: demo

The database service can be started by executing e.g:

```
C:\Adabas D 13.01\bin>x_start mydb
```

3 The technical environment

This chapter deals with the technical prerequisites that are required to conduct an OpenOffice.org automation with Open Object Rexx.

3.1 The Scripting Language Open Object Rexx

In the following provides the evolutionary history of Open Object Rexx and shows some features of this scripting language.

3.1.1 History

Open Object Rexx (ooRexx) has its origin in Rexx, a scripting programming language, created by the IBM engineer Mike F. Cowlishaw in 1979. Its purpose was to replace EXEC II batch language for IBM's mainframes. Over the years Rexx became the standard scripting language for all IBM's operating systems. [Flat06] p.1

In 1988 a group of IBM engineers started their work to design an object oriented extension of Rexx. One strong requirement was the backward compatibility with Rexx, in order that previously developed Rexx programs are still able to work. In 1997 the commercial version of this project, called Object Rexx, was published. [Flat06] p.4

After IBM had announced to release Object Rexx under the Common Public License in 2004, the project's source code was handed over to the REXX Language Association[2]. On February, 22, 2005 the first public release of Open Object Rexx was eventually announced. [Wiki06-1]

[2] „The Rexx Language Association (RexxLA) is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language.“ [RexxLA06]

3.1.2 Overview

This section gives a short overview about the main features of Open Object Rexx.

- **An English-like language**:

  Rexx's command names are oriented to the common English language. Instructions, such as SAY, PULL, DO, END try to ensure an easy learning of this scripting language.

- **Fewer rules**:

  Rexx is not case sensitive, one command can stretch across multiple lines and even keywords (e.g. the above mentioned) are only reserved in context.

- **Interpreted, not compiled**:

  Rexx is, in contrast to Java, an interpreted language and it does not have to be compiled by a compiler.

- **Many useful built-in functions and methods:**

  Several, very useful functions and methods are available.

- **Typeless variables**:

  In Rexx you do not have to declare the type of a variable.

- **Powerful string handling:**

  Rexx includes a wide range of functionalities for manipulating character strings.

- **Decimal Arithmetic**:

  Rexx based its arithmetic operations unlike most other programming language on the decimal arithmetic. In contrast to the widely used binary arithmetic, this calculation method is more accurate.

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

  If an error exists, Rexx displays a detail explanation of the error.

[ooRx06]

3.1.3 Syntax

The Syntax of ooRexx is not explained in this thesis. Please refer to [Flat06] p.5 ff, which gives a very good introduction to the ooRexx syntax. Furthermore [ooRx06-1]

provides the complete ooRexx reference and a programming guide, which describes the syntax and the concepts of ooRexx in detail.

3.2 The Bean Scripting Framework for Rexx

The Bean Scripting Framework for Rexx (BSF4Rexx) is needed to create the connection between ooRexx and OpenOffice.org, or more precisely its Java interfaces. At the beginning of this chapter, the Bean Scripting Framework (BSF) is explained in general followed by a declaration of the more concrete BSF4Rexx.

3.2.1 BSF

> *„Bean Scripting Framework (BSF) is a set of Java classes which provides scripting language support within Java applications, and access to Java objects and methods from scripting languages.“* [Apac01]

For this thesis the last mentioned case is relevant because we use the scripting language ooRexx to access OpenOffice.org's Java interfaces. A similar example would be to write JSPs[3] in a scripting language while having access to the Java class library.

The BSF architecture consists of two primary components: [Apac02]

- **BSF Manager**

  The BSF Manager is responsible for all the registered scripting execution engines. Additional it maintains the object registry that permits script access to Java objects.

- **BSF Engine**

  The BSF Engine provides an interface that has to be implemented by a scripting language, which wants to use BSF. Through this interface an abstraction away from the specific scripting language capabilities is reached. As a consequence a generic handling of script execution and object registration, within the context of the scripting language engine, is provided.

[3] JSP stands for Java Server Pages, which is a Java technology that allows to create e.g. HTML content dynamically.

There are already several BSF Engines existing and each of them supports a specific scripting languages such as Javascript, NetRexx, Python, Tcl, XSLT Stylesheets. However, in order to use ooRexx an own BSF engine is needed, called BSF4Rexx. [Apac01]

3.2.2 BSF4Rexx

BSF4Rexx is the Bean Scripting Language for Rexx. It allows any Java program to invoke Rexx and, the other way round, Rexx scripts are able to get access to Java classes and can communicate with Java objects.

3.2.2.1 History

The development of BSF4Rexx started after Peter Kalender, a student of the University of Essen, had carried out a proof of concept in the course of a seminar paper assigned by Prof. Flatscher. Further development followed by Prof. Flatscher until the first version of BSF4Rexx called "**Essener Version**" was born in 2001. This version was only able to enable Java to execute Rexx Code. [Flat06-1] p.3

In 2003 the "**Augsburger Version**" was published. In this version it was possible for the first time to access Java Classes, objects and methods from ooRexx. This is actually the functionality we need to establish a connection to OpenOffice.org. [Flat06-1] p.4

The latest version is the "**Vienna Version**", which is still enhanced by Prof. Flatscher. Since this version strict Java type definitions are not necessary anymore and some handy tools were added such as new functions for working with OpenOffice.org. The latest release of the Vienna Version is 2.6 and can be downloaded at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/.

3.2.2.2 Usage

The next short examples show how to get the BSF4Rexx support within an ooRexx script.

```
/* print java version using BSF.CLS */
```

```
s = bsf.loadClass('java.lang.System')
jv = s~getProperty('java.version')
say "Used java version: "jv

::requires BSF.CLS /* get BSF support */
```

Snippet 3.1 – Get the BSF4Rexx support.

Snippet 3.1 prints the version of the currently used Java Runtime Environment. The last command is a requires directive which calls `BSF.CLS`.

`BSF.CLS` provides methods, such as `bsf.loadClass()`, that give us the possibility to access Java. With the use of `bsf.loadClass()` a Java class object can be addressed, in this case a reference of the `java.lang.System` class object is made. The variable "`s`" represents a Java object now.

Looking at the Java API[4] you can see that the `java.lang.System` class provides several methods, which can be called now by simply sending a message with the name of the desired method to the Java object `s`.

The method `getProperty('java.version')` returns a string, which can be printed using the ooRexx method `SAY`.

All the following snippets in this thesis, including Snippet 3.2, use the directive `::requires UNO.CLS` instead of `::requires BSF.CLS`. The reason for this is, that `UNO.CLS` provides a lot of useful methods for working with OpenOffice.org and eventually calls `BSF.CLS` itself.

```
/* open an empty Writer document using UNO.CLS */

componentLoader = UNO.createDesktop()~XDesktop~XComponentLoader

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

::requires UNO.CLS   /* get the UNO support including BSF.CLS */
```

Snippet 3.2 – Get an empty Writer document.

[4] The Java API is available on http://java.sun.com/j2se/1.5.0/docs/api/

Snippet 3.2 opens an empty Writer document. The important thing for this snippet to notice is that `UNO.createDesktop()` is a method provided by `UNO.CLS`. The returned service object supports interfaces, which can be easily retrieved by simply sending the name of the interface to the object (e.g. `~Xdesktop`).

Moreover, important to notice is, that the code runs on all platforms unchanged.

In conclusion UNO.CLS provides a lot of useful standard routines, which make your code much shorter, while working with OpenOffice.org, as shown in [Burg06] p22 f. Additionally, in the context of OpenOffice.org automation, the code using ooRexx and BSF4Rexx including UNO.CLS is more concise than using Java directly.

3.3 OpenOffice.org

Mission statement:

> „*To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format.“* [Open06-1]

3.3.1 History

OpenOffice.org has its seeds in StarOffice which was developed by a German company called StarDivision in the mid 80s. In 1999 StarOffice was acquired by Sun Microsystems and some months later StarOffice 5.2 was released free of charge. In 2000 Sun Microsystems made the source code available for the public and a new project was born known as OpenOffice.org. [Wiki06] [Open06-1]

Actually StarOffice still exists as the commercial office suite of Sun Microsystems built on OpenOffice.org's source code. Additionally it includes some licensed-in, third-party technologies such as certain fonts, an extensive ClipArt Gallery, Flash export feature, etc. [StarO1]

In October 2005 OpenOffice.org 2.0 was formally released[5]. This new version was developed with aims of better performance with improved speed and lower memory usage, greater scripting capabilities and better integration. Furthermore a new database front end was designed. Even though it was possible to access data sources with earlier versions, OpenOffice.org 2.0 provides an own application called *Base*[6] for that purpose. [Wiki06]

The latest version at the time of writing is OpenOffice.org 2.1 which was released in December 12th 2006.

[5] The first beta version was already released in 2003.

[6] See *3.3.2. Overview* on page 21 for more information about Base.

3.3.2 Overview

OpenOffice.org 2.0 is a multi platform[7] office suite, available in many different languages. It is an open source product and free available including the source code under the GNU Lesser General Public License (LGPL). One of the project aims is to provide an alternative to the market dominating office suite Microsoft Office. For that reason it supports most of the file formats found in Microsoft Office and many other applications. [Wiki06]

However OpenOffice.org's native file format is the vendor-neutral OASIS[8] OpenDocument file format, which is based on XML. [Open06-1]

OpenOffice.org comprises the following applications: [Open06-2]

- **Writer:** The word processor (similar to Microsoft Word). The bachelor paper [Hinz06] deals with the automation of the Writer application.
- **Calc:** The spreadsheet program (similar to Microsoft Excel). The bachelor paper [Prem06] deals with the automation of the Calc application.
- **Impress:** The presentation program (similar to Microsoft Power Point).
- **Draw:** The vector graphics editor.
- **Math:** A tool for creating and editing formulas.
- **Base:** (similar to Microsoft Access) Base is a database application. Since this thesis is about Data Access, this application will be described in more detail.

  With Base you can create and modify tables, forms, queries and reports either using an external database or the built in HSQL database engine (HSQLDB). Moreover it supports flat file formats such as CSV and several ad-

[7] Multi platform means that it can be deployed on multiple system platforms such as Microsoft Windows, GNU/Linux, Sun Solaris, Mac OS X and FreeBSD [OpenSR].

[8] OASIS stands for “Organization for the Advancement of Structured Information Standards”. It is a global consortium that drives the development, convergence, and adoption of e-business standards. [OASIS06].

dress books like MS Outlook, Mozilla and LDAP[9] which can act as data sources. [Open06-2]

Like the previously mentioned applications Base also has a similar look and feel to Microsoft's counterpart MS Access. Both have graphical interfaces and wizards to provide an easy to use database application.

HSQLDB, Base's integrated database, is a relational database management system written in Java.

3.3.3 Architecture

This chapter describes the architecture of OpenOffice.org, which is needed to know in order to understand how the automation works.

3.3.3.1 UNO – The Base Component Technology

UNO is an abbreviation for Universal Network Object, which is the interface based component model of OpenOffice.org. So OpenOffice.org consists of several UNO components, whereas their interfaces are described in an IDL[10] module. One specific application within OpenOffice.org, for example, the word processor Writer, is simply a collection of different UNO components. These components are reusable, which means that a UNO component that is used in the Writer application can also be a part of the Calc application as shown in illustration 3.1. [Flat05] p.4

[9] LDAP stands for Lightweight Directory Access Protocol and is an Internet protocol, which is used by email applications and other programs to look up information from a server.

[10] IDL stands for Interface Description Language. See [ApiO03] for the OpenOffice.org IDL reference.

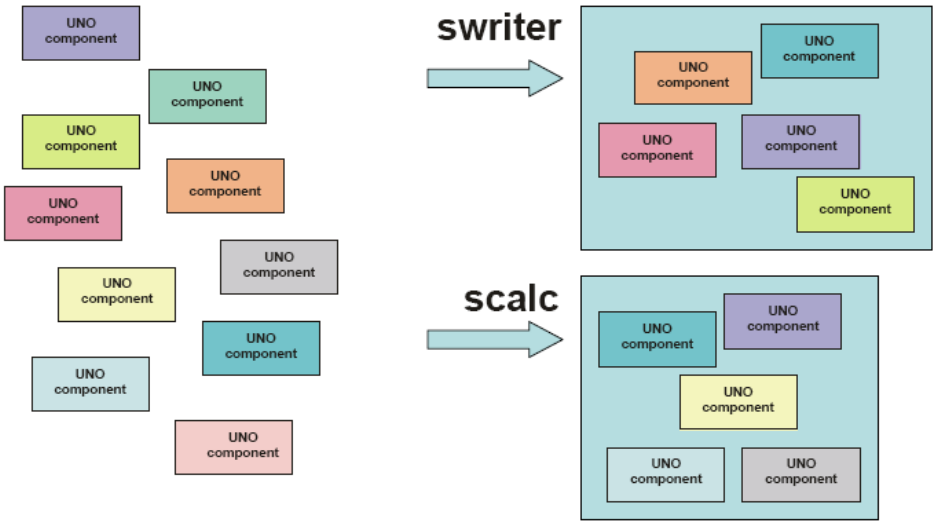

Illustration 3.1: Configuring OpenOffice.org Applications from UNO components. [Flat05] p.5

The communication between the UNO components is performed by the UNO remote protocol (urp), which uses TCP/IP sockets by default. As a consequence it is possible to run OpenOffice.org as a client-server application over a network on different machines. [Flat05] p.6

3.3.3.2 UNO Service Components

In this section OpenOffice.org's essential concept of services, interfaces, properties and attributes will be explained by using, unless otherwise noted, cutouts of Snippet 4.1.

Each UNO component can be considered as a service which provides additional services, interfaces and properties. To create such services, the Service Manager is needed. [Burg06] p.17

3.3.3.2.1 Service Manager and Service Objects

> *„UNO introduces the concept of service managers, which can be considered as factories that create services.“* [Deve05] p.36

The following cutout creates a Service Manager. In more detail the method `UNO.connect`, provided by UNO.CLS[11], is used to get the office component context. Using the method `getServiceManager()` of the component context, the Service Manager can be retrieved. [Deve05] p.37

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

With the help of this Service Manager (`XMcf`) it is possible to create instances of service components, which offer you access to the complete office functionality available through the API. The creation of an instance is accomplished by using the Service Manager's method `createInstanceWithContext()`, supplying the fully qualified name of the UNO component and the previously retrieved component context as shown in the following cutout. [Flat05] p.5 f, [Deve05] p.43

```
/* create a intance of the DatabaseContext service */
databaseContext = xMcf~createInstanceWithContext(-
      "com.sun.star.sdb.DatabaseContext", xContext)
```

In the case mentioned in the cutout an instance of the `com.sun.star.sdb.DatabaseContext` is created. Such an instance is called “service object” but it is also termed as “service” or just “object”.

In some examples, especially older ones, you might find instead of `createInstanceWithContext()` the method `createInstance()`, which is used for creating an instance of a service component. However, it is recommended to use the first one because in the method `createInstance()` the component context is not passed, which should be used to fill missing parameters. [Deve05] p.92

3.3.3.2.2 Interfaces

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

E.g. the previously instantiated *`DatabaseContext`* service supports as shown in Illustration 3.2 among others the `XNameAccess` interface.

[11] See *3.2.2.2 Usage* on page 17 for more information about UNO.CLS.

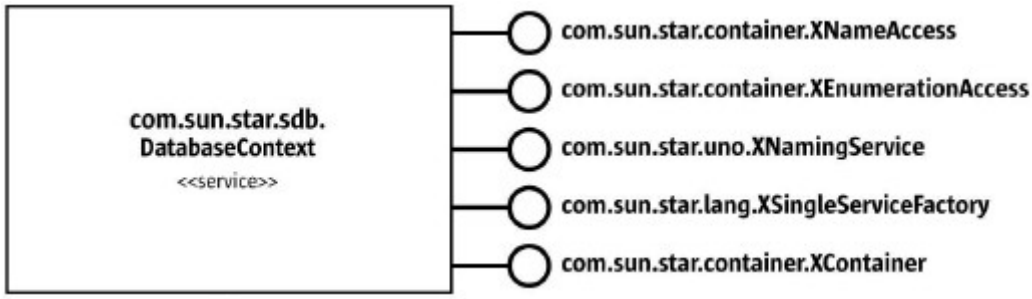

Illustration 3.2: DatabaseContext's interfaces.

An interface can inherit one or more other interfaces, which makes the reuse of interface specifications possible. Moreover multiple inheritance[12] is allowed since OpenOffice.org 2.0. [Deve05] p.39

The following cutout retrieves the `XNameAccess` interface of the *`DatabaseContext`* serivce.

```
/* retrieve the XNameAccess interface of the DatabaseContext */
xNameAccess = databaseContext~XNameAccess
```

You should consider that interfaces possess a capital 'X' in front of their names whereas services do not. Furthermore the instances' names of services and interfaces always begin with a lowercase letter in this thesis. This will help to distinguish between interfaces, services and instantiated objects of them.

The retrieval of an interface has a purpose, certainly. We need it to get access to methods, properties and attributes provided by the interface.

Methods

The following line of code calls a method named `hasByName()`, provided by the *`XNameAccess`* interface of the *`DatabaseContext`* service.

```
say xNameAccess~hasByName("Bibliography") --returns 1 if it exists
```

[12] In this case multiple inheritance refers to the ability to inherit more than one interface

Attributes

Attributes are only available at interfaces and describe additional features of an object. Attributes, unlike properties, can be accessed directly using *get-* and *set-methods*. [Deve05] p.40

For example the `XDocumentDataSource` interface provides an attribute called *DatabaseDocument*. This can easily be retrieved by using the get-method of the attribute as shown in the following cutout taken from Snippet 4.2.

```
/* get OfficeDatabaseDocument service via the 'DatabaseDocument'attribute*/
xOfficeDatabaseDocument = xDocumentDataSource~getDatabaseDocument
```

Properties

Properties are only available via services. Whereas methods and attributes represent features, which are integral parts of the object, properties define abilities that are not considered as an integral or structural part of the object. The object has to support a special interface which allows you to work with properties, which is usually the `XPropertySet` interface. [Deve05] p.41

It provides 2 methods to set or get the required Property:

- `getPropertyValue(aPropertyName, aValue)`
- `setPropertyValue(aPropertyName, aValue)`

Sometimes the value of a property expect an array comprising several properties again as shown in the following cutout taken from Snippet 4.3.

```
props = bsf.createArray(.UNO~propertyValue,1)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "JavaDriverClass"
props[1]~Value = "com.mysql.jdbc.Driver"
xPropertySet~setPropertyValue("Info", props)
```

The method `bsf.createArray()`, provided by BSF.CLS, creates an Java array, with the length of one.

3.4 Overall Concept

In the previous chapters all the single parts, the OpenOffice.org automation is made up of, are explained. Illustration 3.3 gives an idea about the overall concept and the way the parts are composed.

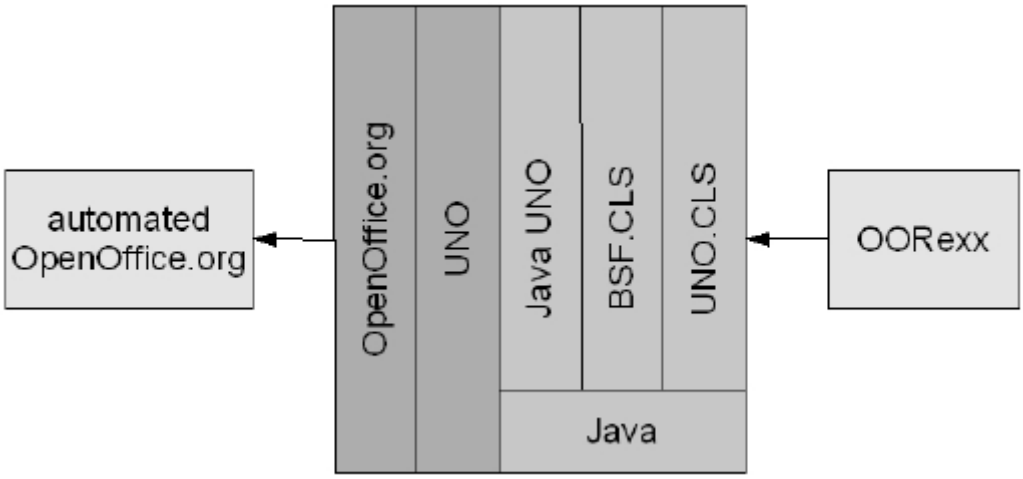

Illustration 3.3: From ooRexx to OpenOffice.org. [Prem06] p.24

The OORexx script calls UNO.CLS, which provides as explained in *3.2.2.2 Usage* on page 17 methods that facilitate working with OpenOffice.org. Within UNO.CLS, BSF.CLS is called which can be considered as a bridge between the scripting language ooRexx and Java.

Next the Universal Network Object (UNO) of OpenOffice.org provide a bridge between OpenOffice.org and a programming language, which is in this case Java.

The result is an OpenOffice.org automation which allows ooRexx to have access to all OpenOffice.org objects.

The advantage of this automation way instead of using Java directly is, that the scripting language ooRexx is very easy to learn and to apply. Additionally UNO.CLS

provides a lot of features which significantly reduce the lines of code compared to Java.

4 Database Access

Beginning with this chapter snippets are provided for several topics beginning, as the title of this thesis suggests, with the Database Access.

Regarding the database access in OpenOffice.org, the most important services are *com.sun.star.sdb.DatabaseContext* and *com.sun.star.sdb.DataSource*.

These services are described blow in more detail.

4.1 DatabaseContext

> *"A data source contains information how to create a connection to a database, such as, which database driver should be used, for which user should a connection be established, etc. The context stores data sources under a given name."* [ApiO03-1]

Therefore the `DatabaseContext`, which is a container for data sources, can be considered as the starting point for applications which aim to connect to data sources already defined in the OpenOffice API. [Deve05] p.830

The following example (Snippet 4.1) prints all data sources which are registered in the `DatabaseContext` (e1_print_registered_data_sources.rex).

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

/* create a instance of the DatabaseContext service */
databaseContext = xMcf~createInstanceWithContext(-
"com.sun.star.sdb.DatabaseContext", xContext)

/* retrieve the XNameAccess interface of the DatabaseContext */
xNameAccess = databaseContext~XNameAccess

/* list all datasource names using the the method getElementNames */
DO n OVER xNameAccess~getElementNames
        say n
END
```

```
say xNameAccess~hasByName("Bibliography") --returns 1 if it exists
say xNameAccess~getByName("Bibliography") --returns the data source

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

Snippet 4.1 - Print registered data sources.

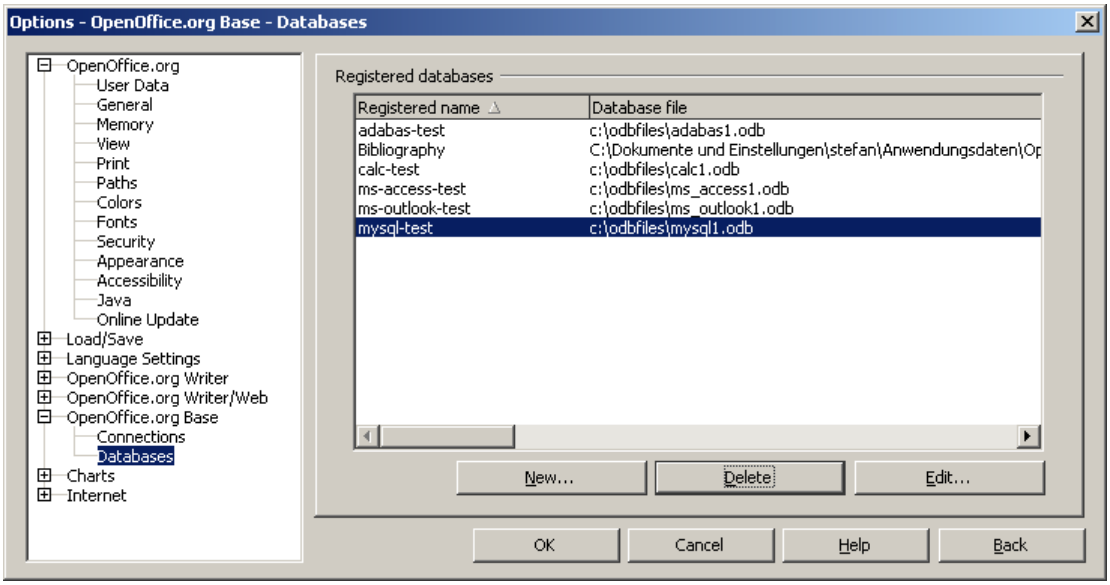

Illustration 4.1: Registered Databases.

The result of this snippet is, among others, a list of all registered data sources, which can also be achieved through the menu *Tools* → *Options...* → *OpenOffice.org Base* → *Databases* of any OpenOffice GUI instance. (cf. Illustration 4.1)

The *DatabaseContext* service holds the registered data sources in a container. To access this data sources the service implements the *com.sun.star.container.XNameAccess* interface which provides the `getElementsNames()` method returning a sequence of all element names in this container. This sequence can be easily enumerated by the ooRexx DO - OVER loop as you can see in the following cutout.

```
/* list all data source names using the the method getElementNames */
DO n OVER xNameAccess~getElementNames
        say n
END
```

Additionally, the *XNameAccess* interface possesses the method `getByName()` to get a specific data source by its name and the method `hasByName()` to retrieve information about the existence of a particular data source. [Deve05] p.53f

```
say xNameAccess~hasByName("Bibliography") --returns 1 if it exists
```

To observe the result of these two methods a say command has been written before. The first method (`hasByName()`) will probably put a "1" on the monitor because it asks about the existence of the Bibliography[13] database, which is a sample database delivered with the OpenOffice.org distribution. If you ask for a data source which does not exist the `hasByName()` method will return a "0".

```
say xNameAccess~getByName("Bibliography") --returns the data source
```

To obtain a specific data source registered in the OpenOffice *DatabaseContext* the `getByName()` method can be used. The previous line of code will cause an output similar to "$Proxy0@b4168ac2" which represents nothing more than a Java-Object. We will use this Object in later snippets to work with the data source (e.g. establish a connection). In order to get a description about the object, you can send the message `toString()` to the object.

4.2 DataSources

> *"The `com.sun.star.sdb.DataSource` service is a factory to establish database connections."* [API03]

It provides several properties (e.g. name, URL, user password) which give information about how to connect to a database and which tables should be displayed. Over its interfaces it is possible to get access to query definitions, forms, reports, etc. [Deve05]

If you want to store any data source into the database context, an OpenDocument Database (odb-File) has to he created at first.

[13] The Bibliography database can be used to create and maintain a bibliography in the OpenOffice.org Writer application. [Open06] p.326 ff

A data source registered at the database context has five main aspects: [Deve05] p.830

- The general information necessary to connect to a data source,
- Settings to control the presentation of tables and queries,
- SQL query definitions,
- Database forms and
- Database reports.

All this information is held in the odb-File. Later there will also be examples which demonstrate how to connect to data sources, which are not registered at the database context. These data sources cannot store e.g. query definitions.

The following snippets will explain the procedure of adding data sources of different databases into the database context.

4.2.1 Adding a Adabas D data source to the database context

In Snippet 4.2 a Adabas D database will be stored into the database context. For general information about the Adabas D database see *2.3.2 Adabas D* on page 11. There you can also find setup instructions, which are necessary to get this snippet work. Since this snippet only saves the information for establishing a connection but actually does not connect to the database, the database service does not need to run.

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

/* retrieve the DatabaseContext and get its XSingleServiceFactory interface */
xSingleServiceFactory = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XSingleServiceFactory

/* create a new generic data source */
dataSource = xSingleServiceFactory~createInstance
```

```
/* setting the necessary data source properties */
xPropertySet = dataSource~XPropertySet
/* Adabas D URL */
xPropertySet~setPropertyValue("URL", "sdbc:adabas::MYDB")
/* force password dialog */
xPropertySet~setPropertyValue(-
        "IsPasswordRequired",.bsf~new("java.lang.Boolean","true"))
/* suggest 'demo' as user name */
xPropertySet~setPropertyValue("User", "demo")

/* get the XDocumentDataSource interface of the data source */
xDocumentDataSource = dataSource~XDocumentDataSource

/* get the OfficeDatabaseDocument service via the 'DatabaseDocument' attribute*/
xOfficeDatabaseDocument = xDocumentDataSource~getDatabaseDocument

/* retrieve the XStorable, xClosable, and XModel interface */
xStorable = xOfficeDatabaseDocument~XStorable
xCloseable = xOfficeDatabaseDocument~XCloseable
xModel = xOfficeDatabaseDocument~XModel

/* register it with the database context */
xNamingService = xSingleServiceFactory~XNamingService
url = uno.ConvertToURL("c:/odbfiles/adabas1.odb")
xStorable~storeAsURL(url,xModel~getArgs)
xNamingService~registerObject("adabas-test", dataSource)
say "database document has been stored to '"url"' !"

/* close database */
xCloseable~close(.true)

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

Snippet 4.2 - Adding a Adabas D data source.

Illustration 4.2: Output of Snippet 4.2.

As a result of running Snippet 4.2 two events should have occurred:

1) There was an additional entry created in the database context called *adabas-test*, which represents the Adabas D database MYDB. You can run Snippet 4.1 to check this.

2) Furthermore, as you can see in the command line, an odb-File was created in the specified folder (c:/odbfiles/).

After opening the newly created file, you have direct access to the Adabas D demo database 'MYDB' (i.e. all change, which will be made affect MYDB) and all the tables in MYDB are shown. If this is not the case assure that the database service is running and the the right password has been typed in ('demo').

The Snippet explained in more detail:

```
xSingleServiceFactory = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XSingleServiceFactory
```

With the aid of the service manager, the *DatabaseConext* can be instantiated, which supports, among others, the *XSingleServiceFactory* interface.

```
dataSource = xSingleServiceFactory~createInstance
```

At first an empty, abstract data source has to be created, using the `createInstance()` method, which is supplied by the *XSingleServiceFactory* interface.

```
xPropertySet = dataSource~XPropertySet
/* Adabas D URL */
xPropertySet~setPropertyValue("URL", "sdbc:adabas::MYDB")
/* force password dialog */
xPropertySet~setPropertyValue(-
        "IsPasswordRequired",.bsf~new("java.lang.Boolean","true"))
/* suggest 'demo' as user name */
xPropertySet~setPropertyValue("User", "demo")
```

This generic data source has to be filled with the appropriate properties[14]. The value of the property *URL* is set to the MYDB database. Furthermore, there is a prefix (sdbc:adabas) which specifies the type of the data source, in this case it is a Adabas D database.

[14] For more information about properties see *3.3.3.2.2 Interfaces* on page 24.

The *IsPasswordRequired* parameter has to be set “true”, because the MYDB database demands a password. Since the corresponding value must be a Java object, `.bsf~new` is used to instantiate a j*ava.lang.Boolean* object with the value *true*.

Finally a user name can be suggested, which will appear e.g. after entering the created data source via the OpenOffice.org Base application.

```
xDocumentDataSource = dataSource~XDocumentDataSource

/* get the OfficeDatabaseDocument service via the 'DatabaseDocument'attribute*/
xOfficeDatabaseDocument = xDocumentDataSource~getDatabaseDocument
```

The *XDocumentDatasource* interface, which is supported by the previously created data source, possesses an attribute called *DatabaseDocument*. This attribute[15] provides access to the *OfficeDatabaseDocument*, which represents a storable odb-document.

```
xStorable = xOfficeDatabaseDocument~XStorable
xCloseable = xOfficeDatabaseDocument~XCloseable
xModel = xOfficeDatabaseDocument~XModel
```

The XStorable, XClosable and XModel interfaces can be retrieved via the *OfficeDatabaseDocument* and are required, as the names suggest, to store, close and to provide the arguments of the model.

```
xNamingService = xSingleServiceFactory~XNamingService
url = uno.ConvertToURL("c:/odbfiles/adabas1.odb")
xStorable~storeAsURL(url,xModel~getArgs)
xNamingService~registerObject("adabasd-test", dataSource)
```

Before the data source can be registered at the database context it has to be stored under a certain URL, which is determined in the “url” variable. The method uno.ConvertToURL provided by UNO.CLS takes an operating system independent URL and converts it into a platform independent filename. During this process it also adds a *file:///* prefix, which is demanded by OpenOffice for absolute file URLs. If the specified folder does not exist, it will be created automatically.

The `storeAsURL()` method is used to store the data source. The second parameter (beside *url*) passed to the method, describes several properties of the docu-

[15] For more information about attributes see *3.3.3.2.2 Interfaces* on page 24.

ment[16]. Afterwards the previously created generic data source can be registered under a certain name (*adabasd-test*) by the `registerObject()s` method.

```
xCloseable~close(.true)
```

Eventually the document is closed using the `close` method provided by the *`XCloseable`* interface.

4.2.2 Adding a MySQL data source to the database context

This snipped registers a MySQL database at the database context. To run it successfully, follow the MySQL setup instructions explained in *2.3.1 MySQL* on page 10.

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

/* retrieve the DatabaseContext and get its XSingleServiceFactory interface */
xSingleServiceFactory = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XSingleServiceFactory

/* create a new generic data source */
dataSource = xSingleServiceFactory~createInstance

/* setting the necessary data source properties */
xPropertySet = dataSource~XPropertySet
/* MySQL URL */
xPropertySet~setPropertyValue("URL", "jdbc:mysql://localhost:3306/test")
/* force password dialog */
xPropertySet~setPropertyValue(-
        "IsPasswordRequired",.bsf~new("java.lang.Boolean","true"))
/* suggest 'stefan' as user name */
xPropertySet~setPropertyValue("User", "stefan")

/* determine the JDBC driver */
props = bsf.createArray(.UNO~propertyValue,1)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "JavaDriverClass"
props[1]~Value = "com.mysql.jdbc.Driver"
xPropertySet~setPropertyValue("Info", props)

/* get the XDocumentDataSource interface of the data source */
```

[16] More precisely it is a sequence of PropertyValue, which transports the "where to" and the "how" of the storing procedure [Deve05] p.833

```
xDocumentDataSource = dataSource~XDocumentDataSource

/* get the OfficeDatabaseDocument service via the 'DatabaseDocument' attribute*/
xOfficeDatabaseDocument = xDocumentDataSource~getDatabaseDocument

/* retrieve the XStorable, xClosable, and XModel interface */
xStorable = xOfficeDatabaseDocument~XStorable
xCloseable = xOfficeDatabaseDocument~XCloseable
xModel = xOfficeDatabaseDocument~XModel

/* register it with the database context */
xNamingService = xSingleServiceFactory~XNamingService
url = uno.ConvertToURL("c:/odbfiles/mysql1.odb")
xStorable~storeAsURL(url,xModel~getArgs)
xNamingService~registerObject("mysql-test", dataSource)
say "database document has been stored to '"url"' !"

/* close database */
xCloseable~close(.true)

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

Snippet 4.3 - Adding a MySQL data source.

This snippet contains almost the same code as Snippet 4.2, shown before. However, there are some little changes and additional lines of code:

```
xPropertySet~setPropertyValue("URL", "jdbc:mysql://localhost:3306/test")
```

At first it is obvious that the URL to the data source is different as well as the prefix. The MySQL database “test”, delivered with the XAMPP distribution is used.

Here the URL `sdbc:mysql:jdbc:localhost:3306/test` is used, which creates a connection of the type “`mysql`”.

The URL `jdbc:mysql://localhost:3306/test` could also be used here, but creates a connection of the type “`jdbc`”.

Actually both URLs initiate a JDBC bridge to the MySQL database. However, the first mentioned one cares for some MySQL particularities concerning parameter handling whereas the second one does not. Using the URL for the connection type “`jdbc`” will cause problems, when accessing the data source inside a form (Snippet 6.1 on page 67).

```
props = bsf.createArray(.UNO~propertyValue,1)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "JavaDriverClass"
props[1]~Value = "com.mysql.jdbc.Driver"
xPropertySet~setPropertyValue("Info", props)
```

In order to connect to a MySQL database an additional property called *JavaDriverClass* is needed. In this property a driver required for connecting to MySQL has to be specified. This property needs to be passed, packed in a Java-Array, to the *Info* property.

4.2.3 Adding other data sources to the database context

Beside the above mentioned databases (Adabas D and MySQL) it is possible to register several other data sources at the database context. Table 4.1 shows the values, the property *URL* has to be set, in order to add the corresponding data source to the database context.

Data Source	***URL***
Microsoft Access	**sdbc:odbc:**<Name of a data source defined in the system>
OpenOffice.org Calc (read only access)	**sdbc:calc:**<file URL>
Microsoft Outlook address book (read only access)	**sdb:address:outlook**
Mozilla address book	**sdbc:address:mozilla**
Derby	**jdbc:derby:**<database URL> (e.g.: jdbc:derby:C:\download\MyDbTest)
HSQL	**jdbc:hsqldb:hsql://**<database> (e.g.: jdbc:hsqldb:hsql://localhost/xdb)

Table 4.1: Data Source URLs.

The reader might ask why it is so important to register a data source at the database context. Actually you can also connect data sources, which are not part of the database context. However, it is easier to connect to the registered one, because all the information required to connect is already stored in such a data source. Furthermore Forms (see *6 Forms* on page 60) have only access to data sources stored in the database context.

4.3 Connections

> *"A Connection is an open communication channel to a database."* [Deve05] p. 852

Connections are used to execute statements, which again return results sets (see *4.4 Manipulate and Query Data* on page 43).

In this thesis two different ways to establish a connection to a data source will be explained:

- Connection through a registered data source[17],
- Connection using the *DriverManager*.

In the following an example for each of this possibilities is presented.

4.3.1 Connection through a registered data source

If the data source, you want to connect to, is registered at the database context, there is, beside the simple **non-interactive** login procedure, the additional opportunity of an **interactive** login procedure. It should be used if a database needs a login and you do not want to hard code the user and the password. [Deve05] p.854

The following snippet contains both login procedures, whereas the non-interactive one is not in use. However, you can easily switch between these login procedures by changing the place of the comments.

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

/* retrieve the DatabaseContext and get its XNameAccess interface */
xNameAccess = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XNameAccess

/* we use the mysql-test data sorce */
dataSource = xNameAccess~getByName("mysql-test")

/************ interactive login ************/
```

[17] See *4.2 DataSources* on page 31 for how to register a data source at the database context.

```
/* create an InteractionHandler and get its XInteractionHandler interface */
interactionHandler = XMcf~createInstanceWithContext(-
        "com.sun.star.sdb.InteractionHandler", xContext);
xInteractionHandler = interactionHandler~XInteractionHandler

/* query for the XCompletedConnection interface of the data source */
xCompletedConnection = dataSource~XCompletedConnection

/* connect with interactive login */
xConnection = xCompletedConnection~connectWithCompletion(xInteractionHandler);


/************ non-interactive login ************/
/*
/* query for the XDataSource interface of the data source */
xDataSource = dataSource~XDataSource

/* simple way to connect - hard code (usr,pw) */
xConnection = xDataSource~getConnection("stefan","apple");

*/


say "Connection created!"

/* get the XClosable interface and close the connection */
xConnection~XCloseable~close
say "Connection closed!"

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

Snippet 4.4 – Connection through a data source.

Illustration 4.3: Output of Snippet 4.4.

The Snippet explained in more detail:

```
dataSource = xNameAccess~getByName("mysql-test")
```

After making an instance of the *DatabaseContext* and retrieving its *XNameAccess* interface we have access to the container the data sources are registered at. With the aid of the `getByName()` method we can retrieve the associated data source as already mentioned in Snippet 4.1 on page 30.

interactive login:

```
interactionHandler = XMcf~createInstanceWithContext(-
        "com.sun.star.sdb.InteractionHandler", xContext);
xInteractionHandler = interactionHandler~XInteractionHandler

/* query for the XCompletedConnection interface of the data source */
xCompletedConnection = dataSource~XCompletedConnection

/* connect with interactive login */
xConnection = xCompletedConnection~connectWithCompletion(xInteractionHandler);
```

To use the interactive login the *XCompletedConnection* interface of the data source has to be retrieved. This interface provides the method `connectWithCompletion()`. This method is passed the main interface of the previously created *InteractioHandler* service. As a result the following window will be displayed, while running the snippet:

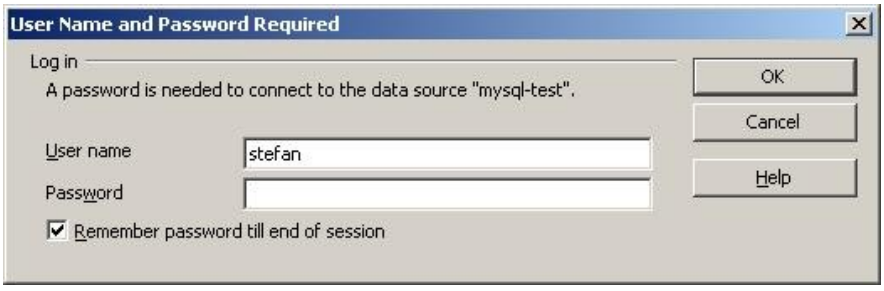

Illustration 4.4: Database Login.

After providing the correct login data a *XConnection* is returned.

non-interactice login:

```
/************ non-interactive login ************/
/*
/* query for the XDataSource interface of the data source */
xDataSource = dataSource~XDataSource

/* simple way to connect - hard code (usr,pw) */
```

```
xConnection = xDataSource~getConnection("stefan","apple");

*/
```

For the non-interactive login, instead of the *XCompletedConnection*, we have to retrieve the *XDataSource*, which provides the `getConnection()` method. There, user name and password are required to receive the *XConnection*.

```
xCloseable = xConnection~XCloseable~close
```

Finally the connection should be closed to release all resources.

4.3.2 Connecting using the DriverManager

If you want to connect to a data source which is not registered at the database context the *DriverManager* service can be used.

```
/* get the service manager */
xContext = UNO.connect()
xMcf = xContext~getServiceManager

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

/* first create the database URL */
url = "jdbc:mysql://localhost:3306/test"

/* create property values for user and password */
props = bsf.createArray(.UNO~propertyValue,3)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "user"
props[1]~Value = "stefan"
props[2] = .UNO~PropertyValue~new
props[2]~Name = "password"
props[2]~Value = "apple"
props[3] = .UNO~PropertyValue~new
props[3]~Name = "JavaDriverClass"
props[3]~Value = "com.mysql.jdbc.Driver"

/* create the connection to mysql */
xConnection = xDriverManager~getConnectionWithInfo(url, props)

say "Connection created by the DriverManager!"

/* get the XClosable interface and close the connection */
xCloseable = xConnection~XCloseable~close
```

```
say "Connection closed!"

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

Snippet 4.5 – Creating a connection using the DriverManager.

The Snippet explained in more detail:

```
xDriverManager = xMcf~createInstanceWithContext(-
        "com.sun.star.sdbc.DriverManager",xContext)~XDriverManager
```

Firstly we have to create an instance of the XDriverManager.

```
url = "jdbc:mysql://localhost:3306/test"

/* create property values for user and password */
props = bsf.createArray(.UNO~propertyValue,3)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "user"
props[1]~Value = "stefan"
props[2] = .UNO~PropertyValue~new
props[2]~Name = "password"
props[2]~Value = "apple"
props[3] = .UNO~PropertyValue~new
props[3]~Name = "JavaDriverClass"
props[3]~Value = "com.mysql.jdbc.Driver"

/* create the connection to mysql */
xConnection = xDriverManager~getConnectionWithInfo(url, props)
```

To receive the *XConnection* the `getConnection()` method can be used, if there is no information, except of the URL, needed to be provided to connect to the data source. However, in the case of the MySQL database *test*, we have to use the method `getConnectionWithInfo()`, which we supply with additional data such as the user name, password and the Java driver class.

4.4 Manipulate and Query Data

There are two possibilities to manipulate data in a database and to set up queries to get a result set.

1. The “statement” object.

2. The “com.star.sdb.RowSet” service.

The next two chapters provide one example for the he “statement” object and one for the “com.star.sdb.RowSet” service.

4.4.1 The statement object

Generally, the procedure to communicate with a database using SQL statements is divided in 4 phases: [Deve05] p.869

1. Get a connection object[18], and
2. Let the connection create a statement.
3. This statement executes a query or an update command. Depending on the command an appropriate method has to be used.
4. If the statement returns a result set, it can be processed.

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

/* retrieve the DatabaseContext and get its XNameAccess interface */
xNameAccess = xMcf~createInstanceWithContext(-
      "com.sun.star.sdb.DatabaseContext", xContext)~XNameAccess

/* we use the "mysql-test" datasorce */
dataSource = xNameAccess~getByName("mysql-test")

/************ non-interactive login ************/
/* query for the XDataSource interface of the data source */
xDataSource = dataSource~XDataSource
/* simple way to connect - hard code (usr,pw) */
xConnection = xDataSource~getConnection("stefan","apple");

/* the connection creates a statement */
xStatement = xConnection~createStatement



/* execute Updates on the database - the basic way*/

/* create the 'product' table and fill it with data */
xStatement~executeUpdate(-
```

[18] For how to get an connection object see *4.3 Connections* on page 39.

```
        "create table product (name varchar(30) primary key, price float)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('Monitor 17inch', 214.90)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('Monitor 19inch', 419.50)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('Monitor 21inch', 759.90)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('Harddisc 160MB', 67.00)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('Sound Card', 43.90)")
xStatement~executeUpdate(-
        "insert into product (name, price) values('SDRAM 1024MB', 156.90)")

say "table 'product' created"

/* create the 'customer' table */
xStatement~executeUpdate(-
        "create table customer (cid int primary key auto_increment, "-
        "firstname varchar(30), lastname varchar(30), birthdate date)")


/* execute Updates on the database - using a prepared statement */

/* create a prepared statement for making inserts in the table customer */
insertCustomerStatement = xConnection~prepareStatement(-
        "insert into customer (firstname, lastname, birthdate) values(?,?,?)")
/* fill the 'customer' table with data */
call insertData insertCustomerStatement, "Tyler", "Durden", "1973-7-2"
call insertData insertCustomerStatement, "Jacob", "Fuller", "1946-9-3"
call insertData insertCustomerStatement, "Mickey", "Knox", "1976-3-17"
call insertData insertCustomerStatement, "Vincent", "Vega", "1963-12-1"
say "table 'customers' created"

/* create the 'sales' table and fill it with data */
xStatement~executeUpdate(-
        "create table sales(sid int primary key auto_increment, cid int,"-
        "name varchar(30), quantity int)")
insertSalesStatement = xConnection~prepareStatement(-
        "insert into sales (cid, name, quantity) values(?,?,?)")

call insertData insertSalesStatement, "1", "Monitor 21inch", "1"
call insertData insertSalesStatement, "2", "Sound Card", "2"
call insertData insertSalesStatement, "2", "Monitor 17inch", "1"
call insertData insertSalesStatement, "3", "Monitor 21inch", "3"
call insertData insertSalesStatement, "3", "Monitor 21inch", "1"
call insertData insertSalesStatement, "4", "Harddisc 160MB", "1"
call insertData insertSalesStatement, "3", "Monitor 19inch", "5"
say "table 'sales' created"
```

```
say


/* execute Query and retrieve the ResultSet */
xResultSet = xStatement~executeQuery(-
        "select cid, firstname, lastname, birthdate from customer")

say "customer Table: (id | firstname | lastname | date of birth)"
IF xResultSet~isBeforeFirst = 0 THEN
        say "no results!"
ELSE DO
        /* process the ResultSet */
        xRow = xResultSet~XRow
        DO WHILE xResultSet~next <> .false
                cid = xRow~getString(1)
                firstname = xRow~getString(2)
                lastname = xRow~getString(3)
                birthdate = xRow~getString(4)
                say cid "|" firstname "|" lastname "|" birthdate
        END
END

::requires UNO.cls -- get UNO support

/* insert data using a prepared statement */
::ROUTINE insertData
USE ARG preparedStatement, data1, data2, data3
xParameters = preparedStatement~XParameters
xParameters~setString(1, data1)
xParameters~setString(2, data2)
xParameters~setString(3, data3)
preparedStatement~executeUpdate
```

Snippet 4.6 – Manipulate data using the statement object.

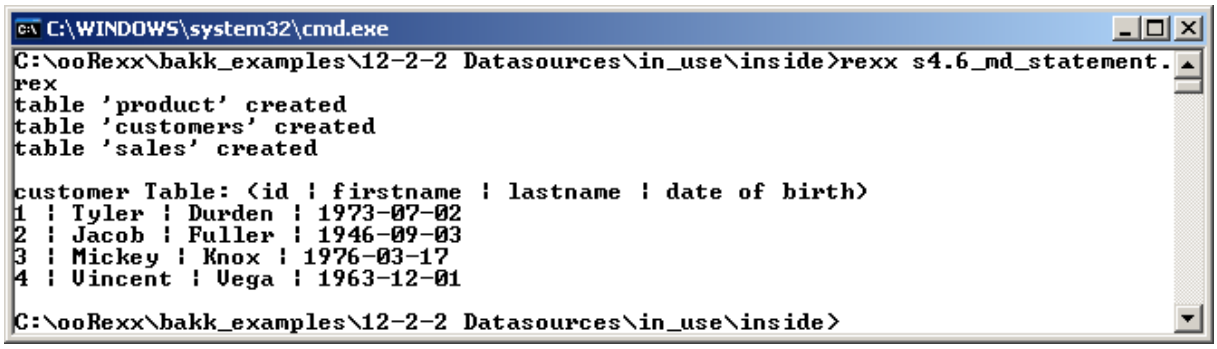

Illustration 4.5: Output of Snippet 4.6.

The snippet explained in more detail:

```
xStatement = xConnection~createStatement
```

The *Connection* is retrieved as explained in Snippet 4.4 on page 40 as well as in Snippet 4.5 on page 43. The statement object is created using the `createStatement()` method of the *Connection*. The statement object can now be used to send SQL statements to the Database Management System (DBMS)[19].

The *XStatement*, which is the main interface of the statement object, provides two methods for sending SQL commands.

- **`executeQuery()`:** use this method for SELECT statements (queries).
- **`executeUpdate()`:** use this method for UPDATE, DELETE, INSERT, DROP, ALTER statements (manipulations).

Now we create some table to build the data structure of a computer shop model[20], which is used in later examples again.

```
xStatement~executeUpdate(-
    "create table product (name varchar(30) primary key, price float)")
```

At first the product table is created. For this purpose we use the `executeUpdate()` method with the appropriate SQL statement.

```
xStatement~executeUpdate(-
        "insert into product (name, price) values('Monitor 17inch', 214.90)")
```

[19] See *2.1 Terms and Definitions* on page 7 for information about DBMS.
[20] An ER-Diagram of the computer shop model is presented in *2.2 Entity Relationship Diagram*.

```
xStatement~executeUpdate(-
    "insert into product (name, price) values('Monitor 19inch', 419.50)")
```

Furthermore we insert some data into the previously created table. As you can see the statement object is reused again, rather than creating a new one for each command. However, each SQL command has to be analyzed and compiled by the DBMS. If you have a lot of similar statements you want to send to the database, it is more efficient to use a "PreparedStatement" object. This object represents a precompiled SQL statement. During the creation of the "PreparedStatement" an SQL statement including parameters is given, which is sent to the DBMS right away where it gets compiled. In a next step it can be adjusted, by the parameters provided, and reused without having the DBMS analyzing and optimizing it again.

```
insertSalesStatement = xConnection~prepareStatement(-
    "insert into sales (cid, name, quantity) values(?,?,?)")
```

The "PreparedStatement" object is created, equally to the statement object, by the Connection. Here a SQL command has to be defined whereas the values, which need to be adjusted later, have to be replaced by a question mark.

To make the call of a prepared statement very easy a method called `insertData()` is written.

```
::ROUTINE insertData
USE ARG preparedStatement, data1, data2, data3
xParameters = preparedStatement~XParameters
xParameters~setString(1, data1)
xParameters~setString(2, data2)
xParameters~setString(3, data3)
preparedStatement~executeUpdate
```

At first we ask for the *`XParameters`* interface of the passed prepared statement. There values can be assigned to each of the parameters, which where previously defined by the question marks. For this purpose we use the `setString()` method with the position number of the question mark and the corresponding data. Finally the prepared statement is sent to the DBMS by method `executeUpdate()`.

```
call insertData insertSalesStatement, "1", "Monitor 21inch", "1"
```

This source code cut out calls the described `insertData()` method.

```
xResultSet = xStatement~executeQuery(-
```

```
        "select cid, firstname, lastname, birthdate from customer")
```

To perform a SELECT statement the `executeQuery()` method has to be used. It returns a result set, which contains the results of the query.

```
IF xResultSet~isBeforeFirst = 0 THEN
        say "no results!"
ELSE DO
        /* process the ResultSet */
        xRow = xResultSet~XRow
        DO WHILE xResultSet~next <> .false
                cid = xRow~getString(1)
                firstname = xRow~getString(2)
                lastname = xRow~getString(3)
                birthdate = xRow~getString(4)
                say cid" | "firstname" | "lastname" | "birthdate
        END
END
```

> *"A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The 'next' method moves the cursor to the next row."* [ApiO03-2]

The first thing we do is checking if the result set is empty (i.e. the query did not deliver any results). If there is at least one entry the cursor of the result set has to have its position before the first entry. Otherwise the result set must be empty. We use the `isBeforeFirst()` method to perform this check.

In a second step we retrieve the *`XRow`* interface which provides access to the data collected in a row. It depends on the position of the cursor, which row the XRow interface represents.

In order that the first row becomes the current row the cursor has to be moved using the `next()` method of the *`XResultSet`*. If the next entry is empty, the value ".false" is returned, otherwise the result is ".true". So we can build a DO-WHILE loop around to traverse all results.

Finally, to get the column values from the current row, we use the `getString()` method. This method needs the number of the column, we want to get the value from.

4.4.2 The RowSet Service

> *The RowSet Service is a client side ResultSet, which combines the characteristics of a Statement and a ResultSet. It acts like a typical bean. Before you use the RowSet, you have to specify a set of properties like a DataSource and a Command and other properties known of Statement.*
>
> *Afterwards, you can populate the RowSet by its execute method to fill the set with data.* [ApiO03-3]

This means the *RowSet* service is a kind of short cut to retrieve data compared to the statement object because you do not have to establish a connection explicitly, create a statement and finally create the result set. Everything can be handled by the *RowSet* service.

```
/* get the service manager */
xContext = UNO.connect()
xMcf = xContext~getServiceManager

/* create RowSet object */
xRowSet = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.RowSet",xContext)~XRowSet
say "RowSet created!"

/* set the properties which are needed to connect to a database */
xPropertySet = xRowSet~XPropertySet
xPropertySet~setPropertyValue("DataSourceName", "mysql-test")
xPropertySet~setPropertyValue("User", "stefan")
xPropertySet~setPropertyValue("Password", "apple")

/* choose the CommandType TABLE and set the command */
xPropertySet~setPropertyValue("Command", "test.sales")
xPropertySet~setPropertyValue("CommandType", box(-
        "int",bsf.getStaticValue("com.sun.star.sdb.CommandType", "TABLE")))

/* now execute the previous specified command */
xRowSet~execute
say "RowSet executed!"

/* process the ResultSet */
say "Results:"
xRow = xRowSet~XRow
DO WHILE xRowSet~next <> .false
        name = xRow~getString(1)
        price = xRow~getString(2)
```

```
        price2 = xRow~getString(3)
        say name"-"price"-"price2
END

say
say "---Infos---"
/* show amount of returned rows*/
say "There are:" xPropertySet~getPropertyValue("RowCount") "rows"
/* show the currently used command */
say "Used Command:" xPropertySet~getPropertyValue("ActiveCommand")
say "-----------"
say

/* destroy the created RowSet */
xComp = xRowSet~XComponent~dispose
say "RowSet destroyed!"

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

Snippet 4.7 - Query data using the RowSet service.

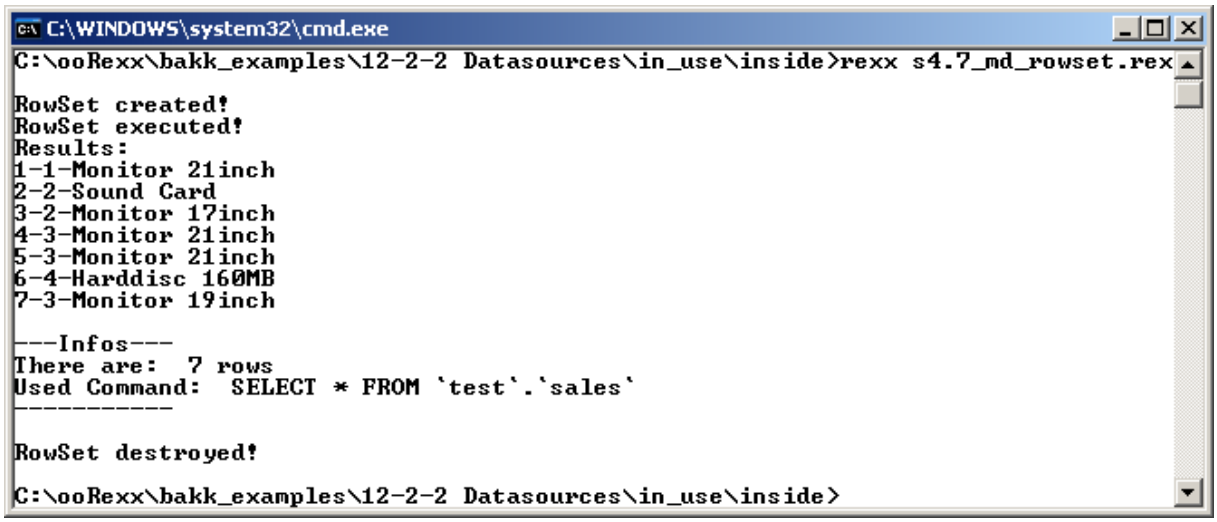

Illustration 4.6: Output of Snippet 4.7.

The snipped explained in more detail:

After making an instance of the *RowSet* service we have to retrieve its *XPropertySet* interface to set some properties such as the name of the data source, the user and the password.

```
xPropertySet~setPropertyValue("Command", "test.sales")
xPropertySet~setPropertyValue("CommandType", box(-
```

```
        "int",bsf.getStaticValue("com.sun.star.sdb.CommandType", "TABLE")))
```

Additionally it is necessary to choose a *CommandType* and declare an appropriate *Command*.

Concerning the *CommandType* you have the choice between three different ones, whereas each expects a different kind of *Command*: (refer to Table 4.2)

CommandType	***Command***	***Comment***
TABLE	a table name	Causes the same result as the SQL statement: `select * from <table>`
QUERY	a predefined query name	See Snippet 5.1 on page 55 for how to create a predefined query.
COMMAND	a SQL command	

Table 4.2: Row Set - Command Types.

In the current snippet the *CommandType* TABLE is used, thus we have to specify a table name. In this case the 'sales' table is chosen. Here it is important to add the name of the database as a prefix: 'test.sales'. The reason for this is that if you connect to an external database, all the tables are stored in a hierarchy under the database name. You can check this by opening the previously created odb-File[21] as shown in figure Illustration 4.7.

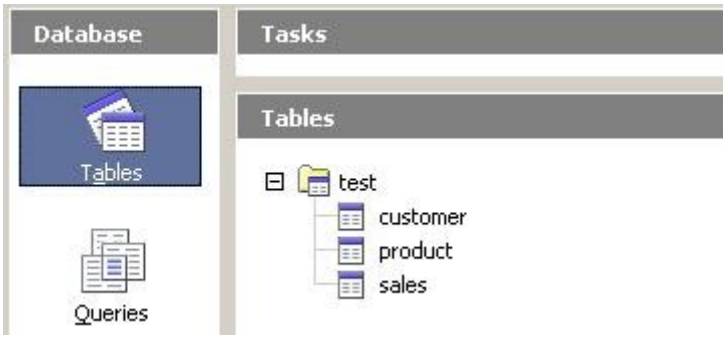

Illustration 4.7: Base Tables.

[21] The appropriate odb-File was created in *Snippet 4.3*

```
xRowSet~execute
```

Now the *RowSet* is ready to be executed and as a consequence of executing, it gets filled with the results. Since the *RowSet* service includes the *ResultSet* service, it acts like a result set and we can use the same methods to process the findings.

```
/* show amount of returned rows*/
say "There are:" xPropertySet~getPropertyValue("RowCount") "rows"
/* show the currently used command */
say "Used Command:" xPropertySet~getPropertyValue("ActiveCommand")
```

The *XRowSet* service comprises properties, which provide some information. The *RowCount* property shows the number of rows. Actually, the cursor has to be positioned after the last entry to get all rows counted. However, since the DO-WHILE loop has been passed through, the cursor is on the last position.

Furthermore the *ActiveCommand* property shows the command, which is currently used. The command line displays as a result of the last line of the cutout's code:

```
Used Command:  SELECT * FROM 'test'.'sales'
```

As Table 4.2 says using the TABLE CommandType is the same as using the '*select * from <table>*' SQL statement. This is correct here. The TABLE command was converted into the appropriate SQL statement.

```
xComp = xRowSet~XComponent~disposexComp
```

The RowSet has to be destroyed.

5 Additional Database Snippet

This chapter shows one additional snippet concerning databases, which does not fit in the chapters above.

5.1 Query Definition

This chapter shows how to insert and execute a *QueryDefinition*. A query definition can be explained as a predefined query. It encapsulates a definition of an SQL statement stored in a database document. The predefined query can be processed multiple times and is even visible in the GUI of the Base application, where it is also possible to execute the query definition. [Deve05] p.834

5.1.1 Store a Query Definition

Snippet 5.1 adds a query definition to the in Snippet 4.3 of page 37 created registered data source “mysql-test”.

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

/* retrieve the DatabaseContext and get its XNameAccess interface */
xNameAccess = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XNameAccess

/* let's use the datasource mysql-test */
dataSource = xNameAccess~getByName("mysql-test")

/*create an empty QueryDefinition and request it's XPropertySet interface*/
xQueryDefinitionsSupplier = dataSource~XQueryDefinitionsSupplier
xQDefs = xQueryDefinitionsSupplier~getQueryDefinitions
xSingleServiceFactory = xQDefs~XSingleServiceFactory
xPropertySet = xSingleServiceFactory~createInstance~XPropertySet

/* define the query */
xPropertySet~setPropertyValue("Command",-
        "SELECT firstname, lastname, product.name "-
        "FROM customer, sales, product "-
        "WHERE customer.cid = sales.cid AND sales.name = product.name")
```

```
xPropertySet~setPropertyValue(-
        "EscapeProcessing",box("BOOL",.true))

/* insert it into the query definition container */
xNameContainer = xQDefs~XNameContainer
queryName = "Query_names+products"
IF xNameContainer~hasByName(queryName) THEN
DO
        say "removing existing query definition...."
        xNameContainer~removeByName(queryName)
END


say "insert Query to the container...."
xNameContainer~insertByName(queryName,xPropertySet)


/* store the database document */

/* retrieve the XDocumentDataSource interface of the data source */
xDocumentDataSource = dataSource~XDocumentDataSource

/* get the attribute 'DatabaseDocument' */
xOfficeDatabaseDocument = xDocumentDataSource~getDatabaseDocument

/* store the database document and with it the query definition */
xStorable = xOfficeDatabaseDocument~XStorable~store
say "storing database document...."

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

Snippet 5.1 – Create and store a query definition.

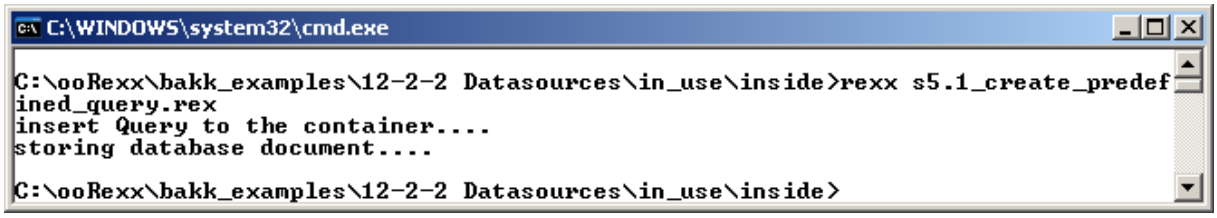

Illustration 5.1: Output of Snippet 5.1.

The snippet explained in more detail:

The query definition will be saved in the underlying database document ("mysql1.odb") of the "mysql-test" data source. For that reason we have to retrieve this data source.

```
xQueryDefinitionsSupplier = dataSource~XQueryDefinitionsSupplier
xQDefs = xQueryDefinitionsSupplier~getQueryDefinitions
xSingleServiceFactory = xQDefs~XSingleServiceFactory
xPropertySet = xSingleServiceFactory~createInstance~XPropertySet
```

The *XQueryDefinition* interface, provided by the data source, is used to receive the *DefinitionContainer* (xQDefs). This container comprises all the query definitions stored so far in the specific data source. Since OpenOffice.org containers generally support the *XNameAccess* interface, it is possible to access each stored query definition or to traverse them using a DO-WILE as explained in Snippet 4.1 on 30.

With the aid of the *XSingleServiceFactory* and its method `createInstance()`, an empty query definition is created. In order to adjust this query definition we need its *XPropertySet* interface.

```
xPropertySet~setPropertyValue("Command",-
        "SELECT firstname, lastname, product.name "-
        "FROM customer, sales, product "-
        "WHERE customer.cid = sales.cid AND sales.name = product.name")

xPropertySet~setPropertyValue(-
        "EscapeProcessing",box("BOOL",.true))
```

Afterwards two properties are set. First the SQL command and second the *EscapeProcessing* to "true" because we do not want the built-in SQL parser to touch the query.

```
xNameContainer = xQDefs~XNameContainer
queryName = "Query_names+products"
IF xNameContainer~hasByName(queryName) THEN
DO
        say "removing existing query definition...."
        xNameContainer~removeByName(queryName)
END
```

```
say "insert Query to the container...."
xNameContainer~insertByName(queryName,xPropertySet)
```

In a next step the query definition has to be added to the DefinitionContainer (xQDefs). In OpenOffice.org containers, which should provide the ability to add and remove elements, support the *XNameContainer* interface. This applies to the DefinitionContainer. The query definition is added using the method `insertByName()` and passing a name and the previously retrieved *XPropertySet* interface, which represents the query definition. Before we append the query, the existence of the chosen name is checked. If there is already a query definition with the same name stored, it will be deleted to avoid an error.

Finally the database document has to be stored. This works the same way as already shown in snippet Snippet 4.3.

5.1.2 Execute a Query Definition

As there are two possibilities to set up a SQL query (see *4.4 Manipulate and Query Data* on page 43), there are also two possibilities to execute a query definition.

- The RowSet service and
- The XCommandPreparation interface of a connection object

The RowSet Service

In order to execute a query definition using the RowSet service refer to *4.4.2 The RowSet Service* on page 50. Set the CommandType to QUERY and provide the name of the query definition.

The XCommandPreparation interface of a connection object

Snippet 5.2 shows how to execute a query definition using the *XCommandPreparation* interface of a connection object.

```
/* get the service manager */
```

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

/* retrieve the DatabaseContext and get its XNameAccess interface */
xNameAccess = xMcf~createInstanceWithContext(-
        "com.sun.star.sdb.DatabaseContext", xContext)~XNameAccess

/* we use the "mysql-test" datasource */
dataSource = xNameAccess~getByName("mysql-test")

/************ non-interactive login ************/
/* query for the XDataSource interface of the data source */
xDataSource = dataSource~XDataSource
/* simple way to connect - hard code (usr,pw) */
xConnection = xDataSource~getConnection("stefan","apple");


/* execute a query definition */
xCommandPreparation = xConnection~XCommandPreparation
xPreparedStatement = xCommandPreparation~prepareCommand(-
        "Query_names+products",-
        bsf.getStaticValue("com.sun.star.sdb.CommandType", "Query"))

xResultSet = xPreparedStatement~executeQuery


xRow = xResultSet~XRow
DO WHILE xResultSet~next <> .false
        firstname = xRow~getString(1)
        lastname = xRow~getString(2)
        product = xRow~getString(3)
        say firstname "|" lastname "|" product
END

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

Snippet 5.2 – Execute a query definition.

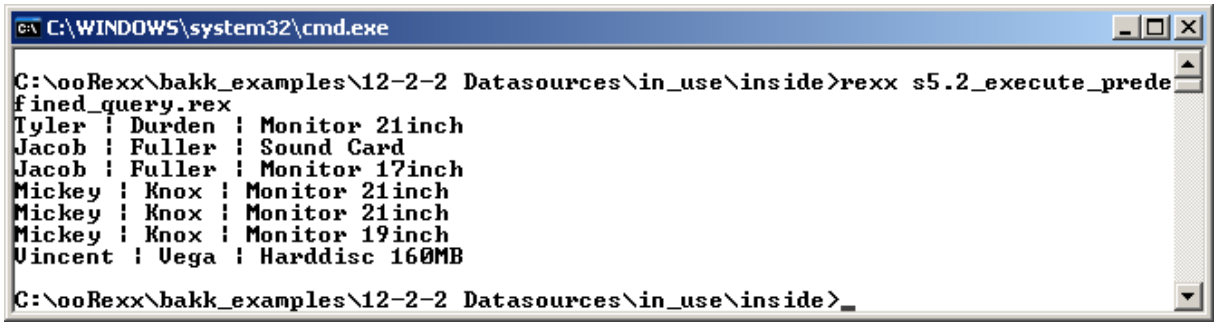

Illustration 5.2: Output of Snippet 5.2

Snippet 5.2 is very similar to Snippet 4.6 page 46. The new lines of code, which are responsible for executing the query definition, are shown in the following cutout.

```
xCommandPreparation = xConnection~XCommandPreparation
xPreparedStatement = xCommandPreparation~prepareCommand(-
        "Query_names+products",-
        bsf.getStaticValue("com.sun.star.sdb.CommandType", "Query"))

xResultSet = xPreparedStatement~executeQuery
```

As the headline of this snippets suggests, we need the *XCommandPreperation* interface of a connection object to execute a predefined query. It provides the method `prepareCommand()` where the query definition's name has to be passed as well as the *CommandType* QUERY. Afterwards the prepared statement can be executed, which returns a result set.

6 Forms

In this chapter a snippet is illustrated which creates a form document. Afterwards the created form is added into to database document.

6.1 Create a New Form

The in Java written Developers Guide Form-Example, provided by the OpenOffice.org SDK[22], has been taken as a pattern for the development of the following Snippet 6.1. For an easier understanding of forms and how to create them Snippet 6.1 was simplified in some aspects compared to the SDK Form-Example.

The following form snippet uses the model of the computer shop described in *2.2 Entity Relation Diagram*. The aim of this snippet is to create a so-called “data aware form”. This means that the user can manipulate the data from a database via this form. The form consists of a master form which gives the information for each customer and a sub form that provides data of all sales carried out by the currently shown customer.

[22] SDK stands for Software Development Kit. The SDK of OpenOffice.org provides several necessary tools and documentation for programming the OpenOffice.org API. It is available at http://download.openoffice.org/2.0.4/sdk.html

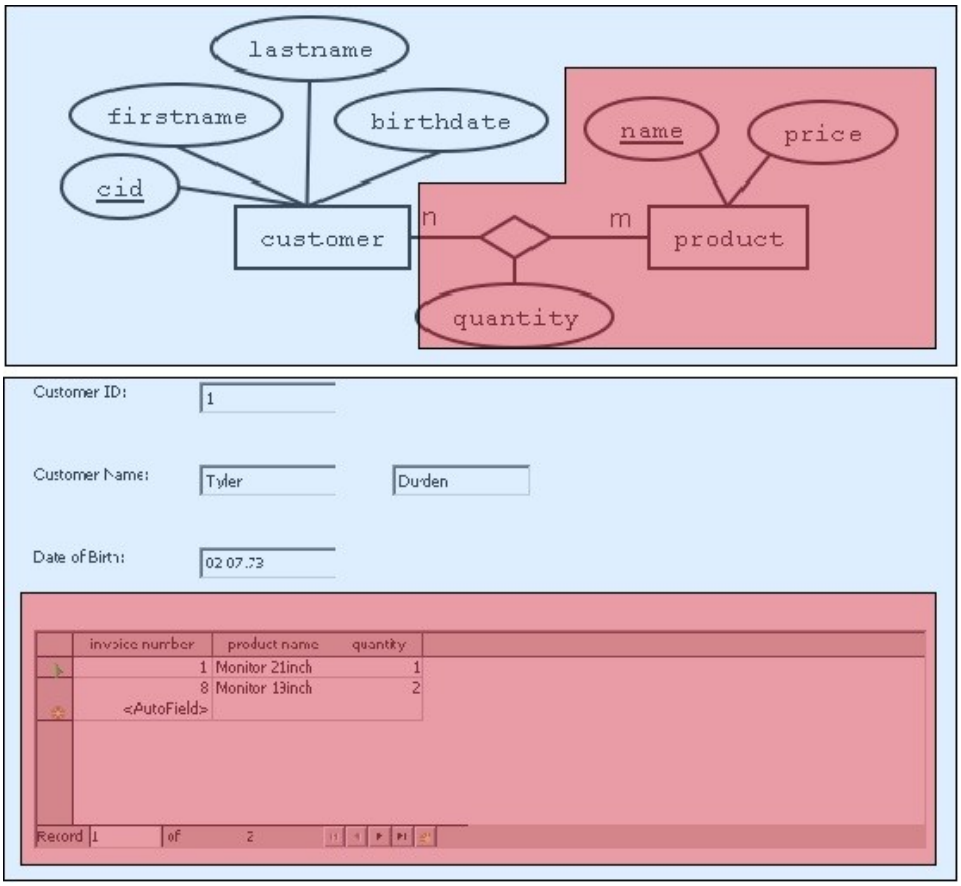

Illustration 6.1: Master Form - Sub Form Relation.

Illustration 6.1 shows how the main form and the sub form are related. The main form is represented by the blue area, the sub form by the red area. The bottom part of illustration 6.1 presents the outcome of the snippet. It is possible to switch between the customers by using the navigation buttons of the “Form Navigation” tool bar pictured in illustration 6.2. Additionally, you can add new customers and sales directly into the database by using the text fields and the table.

Illustration 6.2: Form Navigation Toolbar.

Since Snippet 6.1 uses the data model of the computer shop, the following two snippets need to be run in the specified order, in order to get Snippet 6.1 work.

1. Snippet 4.3
2. Snippet 4.6

```
/* get the service manager */
xContext = UNO.connect() --connect to server and retrieve the XContext object
xMcf = xContext~getServiceManager -- retrieve XMultiComponentFactory

oDesktop = UNO.createDesktop() -- get the UNO Desktop service object

/* get componentLoader interface */
xComponentLoader = oDesktop~XDesktop~XComponentLoader

/* create a new Writer file */
url = "private:factory/swriter"
xComponent = xComponentLoader~loadComponentFromURL(-
        url, "_blank", 0, .UNO~noProps)

/* Save some objects which are often used in the following routines into the
directory object '.local'. So they can be accessed from the whole Rexx program
like global variables */
.local~xMcf = xMcf
.local~xContext = xContext
.local~xComponent = xComponent


/************************** create form components ************************/

label1 = createControlAndShape("FixedText", 25, 6, 5, 5)
label1~setPropertyValue("Label", "Customer ID:") --assign the display-text

textfield1 = createControlAndShape("TextField", 25, 6, 35, 5)
textfield1~setPropertyValue("DataField", "cid") --assign to a field in the db
textfield1~setPropertyValue("Name", "cid_textfield") --assign a name

label2 = createControlAndShape("FixedText", 25, 6, 5, 20)
label2~setPropertyValue("Label", "Customer Name:") --assign the display-text

textfield21 = createControlAndShape("TextField", 25, 6, 35, 20)
textfield21~setPropertyValue("DataField", "firstname")
textfield21~setPropertyValue("Name", "firstname_textfield") --assign a name

textfield22 = createControlAndShape("TextField", 25, 6, 70, 20)
textfield22~setPropertyValue("DataField", "lastname")
textfield22~setPropertyValue("Name", "lastname_textfield") --assign a name
```

```
label3 = createControlAndShape("FixedText", 25, 6, 5, 35)
label3~setPropertyValue("Label", "Date of Birth:") --assign the display-text

textfield3 = createControlAndShape("TextField", 25, 6, 35, 35)
textfield3~setPropertyValue("DataField", "birthdate")
textfield3~setPropertyValue("Name", "birthdate_textfield") --assign a name

/************************* bind MasterForm to the database ****************/

/* get the parent of any control model inserted previously to obtain the
   control model of the form, this form was automaticaly created while inserting
   the first control model of a form component.
   The masterForm is like every form component a 'XPropertySet'
*/

masterForm = label1~XChild~getParent~XPropertySet

masterForm~setPropertyValue("DataSourceName", "mysql-test")
masterForm~setPropertyValue("CommandType",  box(-
        "int",bsf.getStaticValue("com.sun.star.sdb.CommandType", "COMMAND")))
masterForm~setPropertyValue("Command", "select * from test.customer")
masterForm~setPropertyValue("Name", "customers_form")

/************************* create sub form (sales) ************************/

/* masterContainer is an XIndexContainer */
masterContainer = masterForm~XIndexContainer
/* create a new form */
salesForm = .xMcf~createInstanceWithContext(-
        "com.sun.star.form.component.DataForm", xContext)
/* insert it into the parentContainer */
masterContainer~insertByIndex(masterContainer~getCount, salesForm )
salesForm~XPropertySet~setPropertyValue("Name", "sales_form")

/************************* bind SubForm to the database *******************/

/* salesFormProps is a XPropertySet */
salesFormProps = salesForm~XPropertySet

salesFormProps~setPropertyValue("DataSourceName", "mysql-test")
salesFormProps~setPropertyValue("CommandType", box("int",bsf.getStaticValue(-
        "com.sun.star.sdb.CommandType", "COMMAND")))

command = "select * from test.sales where cid = :c"
salesFormProps~setPropertyValue("Command",command)


/************************* establish masterForm-SubForm connection**********/
```

```
strArray = bsf.createArray("String.class", 1)
strArray[1] = "cid"
salesFormProps~setPropertyValue("MasterFields", strArray)
strArray[1] = "c"
salesFormProps~setPropertyValue("DetailFields", strArray)


/* now create the grid model and insert it into the salesContainer*/
/* get the XIndexContainer interface of the salesForm
   salesContainer is a XIndexContainer */
salesContainer = salesForm~XIndexContainer
call createControlAndShape "GridControl", 162, 40, 5, 50, salesContainer
gridControl = result

gridControl~setPropertyValue("Name", "sales_table")

call newGridColumn gridControl, "TextField", "sid", "invoice number"

call newGridColumn gridControl, "ListBox", "name", "product name"
productName = result --save the 'product name' column to set some more props

/* initialize the 'product name' ListBox to provide a choice of product names*/
productName~setPropertyValue( "ListSourceType", bsf.getConstant(-
        "com.sun.star.form.ListSourceType","SQL"))
productName~setPropertyValue("BoundColumn", .BSF~new("java.lang.Short","1"))

sListSource = "SELECT product.name, product.name FROM test.product"
strArray[1] = sListSource
productName~setPropertyValue( "ListSource", strArray)

call newGridColumn gridControl, "TextField", "quantity", "quantity"

/* switch to live mode */
call toggleFormDesignMode

say "Form Document created!"

/* save form and close document*/
xComponent~XStorable~storeAsURL(-
        uno.ConvertToURL("c:/odbfiles/shop_form.odt"), .UNO~noProps)
xComponent~XCloseable~close(.true)

::requires UNO.cls -- get UNO support

----------------------------------ROUTINEs-------------------------------------
-------------------------------------------------------------------------------

/* routine switches between the life-mode and the design mode */
::routine toggleFormDesignMode
```

```
URL = bsf.import('com.sun.star.util.URL')

aToggleURL = bsf.createArray(URL, 1)
aToggleURL[1] = URL~new
aToggleURL[1]~Complete = ".uno:SwitchControlDesignMode"

--need an URLTransformer
frameDesktop = .xMcf~createInstanceWithContext(-
        "com.sun.star.util.URLTransformer",.xContext)
frameDesktop~XURLTransformer~parseStrict(aToggleURL)

/* get the XController interface of the controller service */
xController = .xComponent~XModel~getCurrentController~XController
--go get the dispatch provider of it's frame
xDispatchProvider = xController~getFrame~XDispatchProvider

xDispatch = xDispatchProvider~queryDispatch( aToggleURL[1],"", 0 )
xDispatch~dispatch( aToggleURL[1], .UNO~noProps )

------------------------------------------------------------------------

/* creates a control shape, together with a control model, and inserts them
   into the document model */
/*      ARG(1) - name: String sQualifiedComponentName (f.e. TextField)
        ARG(2) - width: int Size
        ARG(3) - height: int Size
        ARG(4) - x: int Position - X-axis
        ARG(5) - y: int Position - Y-axis
        ARG(6) - parent: XIndexContainer xParentForm OPTIONAL
*/
::routine createControlAndShape
use arg arg_name, arg_width, arg_height, arg_x, arg_y, arg_parent
/********** create and initialize the shape **********/
/* let the document create a shape */
xDocAsMultiServiceFactory = .xComponent~XMultiServiceFactory
xControlShape = xDocAsMultiServiceFactory~createInstance(-
        "com.sun.star.drawing.ControlShape")~XControlShape

/* set position and size of the shape */
xControlShape~setSize(.bsf~new(-
        "com.sun.star.awt.Size", arg_width*100, arg_height*100)) --in 100th/mm
xControlShape~setPosition(.bsf~new(-
        "com.sun.star.awt.Point", arg_x*100, arg_y*100))

/* adjust the anchor so that the control is tied to the page */
xPropertySet = xControlShape~XPropertySet
xPropertySet~setPropertyValue("AnchorType", bsf.getConstant(-
        "com.sun.star.text.TextContentAnchorType", "AT_PARAGRAPH"))
```

```
/********** create a control model **********/
/* create the form component (the model of a form control) */
sQualifiedComponentName = "com.sun.star.form.component."arg_name
xControlModel = .xMcf~createInstanceWithContext(-
        sQualifiedComponentName,.xContext)~XControlModel

/* if "ARG(6):xParentForm" is given insert the form component into the
   appropriate location of the form hierarchy
   if the argument is not given the form component is automatically inserted
   into the root form */
IF arg_parent <> "ARG_PARENT" THEN DO
        arg_parent~insertByIndex( arg_parent~getCount, xControlModel )
END

/********** announce the control model to the shape **********/

/* knitt them */
xControlShape~setControl(xControlModel)

/******* insert the shape into the shapes collection of a draw page ********/

/* add the shape to the shapes collection of the document */
xDrawPageSupplier = .xComponent~XDrawPageSupplier
xDrawPageSupplier~getDrawPage~XShapes~add(xControlShape)

/* return the XPropertySet interface of the ControlModel.
   With it it's possible to make several adjustments at the ControlModel */
return xControlModel~XPropertySet

------------------------------------------------------------------------------

/* creates a new Column inside a 'GridControl' */
/*      ARG(1) - container: grid control
        ARG(2) - name: String componentName
        ARG(3) - field: String dataField
        ARG(4) - label: String label
*/
::routine newGridColumn
use arg arg_container, arg_factory, arg_name, arg_field, arg_label
/* query for the container to insert columns into */
xIndexContainer = arg_container~XIndexContainer
/* query for the factory for creating the column models */
xGridColumnFactory = arg_container~XGridColumnFactory

/* create a new column */
newColumn = xGridColumnFactory~createColumn(arg_factory)
xPropertySet= newColumn~XPropertySet
xPropertySet~setPropertyValue("DataField", arg_name)
```

```
xPropertySet~setPropertyValue("Label", arg_field)
/* insert the column */
xIndexContainer~insertByIndex(xIndexContainer~getCount, newColumn)
return xPropertySet --return the created column
```

Snippet 6.1 - Create a form.

Illustration 6.3: Output of Snippet 6.1.

As you can see the snippet is quit long. For that reason the explanation is divided into 5 parts. They reflecting the basic procedure of creating a Form:

1. Create a new Writer document.
2. Create Components inside the Master Form.
3. Bind Forms to the Database.
4. Create Components inside the Sub Form.
5. Switch to the Live Mode.

6.1.1 Create a new Writer document

A form document is basically an ordinary OpenOffice.org text document with form controls (i.e. for example a text field). As a consequence the first step is to create an empty Writer document.

```
oDesktop = UNO.createDesktop() -- get the UNO Desktop service object

/* get componentLoader interface */
xComponentLoader = oDesktop~XDesktop~XComponentLoader

/* create a new Writer file */
url = "private:factory/swriter"
xComponent = xComponentLoader~loadComponentFromURL(-
        url, "_blank", 0, .UNO~noProps)
```

After getting the *XContext* and the *XMultiComponentFactory*, which works the same way as in the previous snippets, the *ComponentLoader* has to be created. Its main interface *XComponentLoader* provides the method `loadComponentFromURL()`, which has to be used for loading existing OpenOffice.org documents or creating new ones. This method needs several parameters whereby the specified attributes create a new and empty Writer document. For more information about creating and loading OpenOffice.org documents refer to [Aham05] p.27 ff.

```
.local~xMcf = xMcf
.local~xContext = xContext
.local~xComponent = xComponent
```

The *XMultiComponentFactory*, the *xContext* and the *XComponent* loader has to be used very often within this snippet. The first two mentioned are especially needed for creating instances of services. In order to use them even in routines without importing them each time, the “Directory-object” .local is used. This directory initially comprises some objects such as the output object (represents the default output stream), but it can also pick up new objects. For storing a new object there, simply send a message with its name to the Local Environment Object and afterwards it can be accessed by a dot followed by the specified name. E.g. the *xComponent* interface can be accessed by `.xComponent` from now on.

6.1.2 Create Components Inside the Master Form

After a new Writer document has been created we can add form components. To facilitate the creation of form components a routine called `createControlAndShape()` was written. This routine only needs the type of the component (e.g. “TextField”), the size and the position and adds the specified form component afterwards to the document. Now the **`createControlAndShape()`** method will be described in more detail:

The basic procedure for the creation of such a form component can be divided in four steps: [Deve05] p.929

1. Create and initialize a shape,

2. Create a control model,

3. Announce the control model to the shape, and

4. Insert the shape into the shapes collection of a draw page.

Create and initialize a shape:

```
xDocAsMultiServiceFactory = .xComponent~XMultiServiceFactory
xControlShape = xDocAsMultiServiceFactory~createInstance(-
        "com.sun.star.drawing.ControlShape")~XControlShape
```

To create a new shape inside the document the *MultiServiceFactory* service of *XComponent*, which represents the Writer document, is needed. With the use of this factory a new instance of the *ControlShape* service can be created. The control shape is responsible for setting the size and position for a control model (e.g. a text field), which can be bound to the control shape later.

```
/* set position and size of the shape */
xControlShape~setSize(.bsf~new(-
        "com.sun.star.awt.Size", arg_width*100, arg_height*100)) --in 100th/mm
xControlShape~setPosition(.bsf~new(-
        "com.sun.star.awt.Point", arg_x*100, arg_y*100))

/* adjust the anchor so that the control is tied to the page */
xPropertySet = xControlShape~XPropertySet
xPropertySet~setPropertyValue("AnchorType", bsf.getConstant(-
        "com.sun.star.text.TextContentAnchorType", "AT_PARAGRAPH"))
```

To do this the *XControlShape* interface, which is the main interface of the *ControlShape* service, provides the appropriate methods `setSize()` and `setPosition()`. Additionally the property *AnchorType* can be specified, which defines how the content of the control shape (i.e. a text field for example) is attached to its surrounding. The value “AT_PARAGRAPH” sets the anchor of the object at the top left of the paragraph.

Create a control model:

```
sQualifiedComponentName = "com.sun.star.form.component."ARG(1)
xControlModel = .xMcf~createInstanceWithContext(-
        sQualifiedComponentName,.xContext)~XControlModel
```

In the next step the form component, which is specified in the first parameter of the routine call (e.g. “TextField”), is created. There are several types of form components provided in OpenOffice.org. Within this snippets the following types are used:

- FixedText: A text which can be displayed but not edited by the user.
- TextField: Allows the input of a text.
- GridControl: A component which can display data in a table-like way.
- ListBox: This component provides a list of several alternatives values from which a user can make a choice.

Each form component includes the *FormControlModel* service, which follows the model-view-paradigma. For that reason a form component is also a control model.

The **model-view-paradigm** says that for a given element in a document there is exactly one model and an arbitrary number of views. On the one hand the model is stored in the document file and describes how this element looks like in the view and how it behaves. On the other hand the view is a visual representation of the model (i.e. everything the user can see) and only exists if there is an open instance of the document. If a user wants to insert data into a form document (e.g. a text field) in a first step only the view gets notified about this action because it is the view that the user can see and communicate with. In a second step the view is responsible for forwarding the changes to the model. [Deve05] p.922

```
IF arg_parent <> "ARG_PARENT" THEN DO
        arg_parent~insertByIndex( arg_parent~getCount, xControlModel )
END
```

Several forms in one document are organized hierarchically. If there is no information given about the location a form component should be inserted to, it will be automatically added to the root form. E.g. if there is also a sub form present in a document and form components should be inserted there, this has to be specified as the previous cutout shows. In that case the form, in which the form component should be added, has to be passed to the routine as “arg_parent”.

Announce the control model to the shape:

```
xControlShape~setControl(xControlModel)
```

Now the control model of the form component can be assigned to the shape.

Insert the shape into the shapes collection of a draw page:

```
xDrawPageSupplier = .xComponent~XDrawPageSupplier
xDrawPageSupplier~getDrawPage~XShapes~add(xControlShape)
```

Finally, the control shape, which comprises the control model, has to be inserted into the page of the Writer document. For this reason the *DrawPage* service is needed, which actually represents the page containing the drawings. This service can be obtained by the *XDrawPageSupplier* interface of the Writer document. One of its included services provides the *XShapes* interface, which manages the shapes collection of this page. Here it is possible to remove or in this case add shapes. As a consequence, the previously created control shape is added here.

In this state the control model has been inserted and is shown in the document.

```
return xControlModel~XPropertySet
```

The last command in this routine returns the *XPropertySet* interface of the control model. This provides the possibility to make several adjustments to the control model if necessary. This adjustments are e.g. assigning a name or a data field to the control model, which is shown in the following cutout.

```
textfield1 = createControlAndShape("TextField", 25, 6, 35, 5)
textfield1~setPropertyValue("DataField", "cid") --assign to a field in the db
textfield1~setPropertyValue("Name", "cid_textfield") --assign a name
```

Now the `createControlAndShape()` routine can be called as the upper cut out shows. Additionally some properties needs to be set such as the *Name* and the *DataField*. The last mentioned property specifies which data field of the form's result set should be visible in the text field. How to bind a result set to a form will be explained in the next chapter. In this case the data field cid (i.e. the customer ID of the customer table) is shown.

6.1.3 Bind Forms to the Database

In order to make a form in OpenOffice.org data aware it has to be associated with a row set[23]. This is possible because the *DataForm* service, which specifies the form implements the *RowSet* service.

```
masterForm = label1~XChild~getParent~XPropertySet

masterForm~setPropertyValue("DataSourceName", "mysql-test")
masterForm~setPropertyValue("CommandType",  box(-
        "int",bsf.getStaticValue("com.sun.star.sdb.CommandType", "COMMAND")))
masterForm~setPropertyValue("Command", "select * from test.customer")
masterForm~setPropertyValue("Name", "customers_form")
```

At first this is done for the master form, which owns several properties for this purpose. However, in order to get access to this property the master form, which is the root form in the hierarchy of forms in our document, has to be retrieved afore. This form was created automatically as the first form component was created and inserted in the page. Each form component provides the method `getParent()` via its *XChild* interface to get the form in which it resides. As a result, the master form can be retrieved through any form component located there.

The master form is bound to the whole customer table. How to set the properties for an RowSet to execute such a query has already been explained in Snippet 4.7 on page 51. Furthermore for clarity reasons the name “customer_form” is assigned to the master form.

Now the sub form needs to be created.

```
masterContainer = masterForm~XIndexContainer
/* create a new form */
salesForm = .xMcf~createInstanceWithContext(-
        "com.sun.star.form.component.DataForm", xContext)
/* insert it into the parentContainer */
masterContainer~insertByIndex(masterContainer~getCount, salesForm )
```

In order to do this, a new instance of the *DataForm* service has to be created. This form is added to the container of the master form afterwards. The `insertByIndex()` method is used for this purpose whereby the value of the first

[23] Row sets have already been explained in *4.4.2 The RowSet Service* on page 50.

attribute specifies the index of the position inside the container. `masterConatiner~getCount` makes sure that the new form is added to the last position of the container.

This sub form is also bound to the same MySQL database as the master form but the command looks different of course.

```
command = "select * from test.sales where cid = :c"
```

In the sub form the sales made by the customer, who is currently shown in the master form, shall be displayed. Consequently, the command of the sub form selects everything from the sales table where the customer ID (cid) is equal to a parameter called “c”. This parameter will be used to create the connection between the master and the sub form.

```
strArray = bsf.createArray("String.class", 1)
strArray[1] = "cid"
salesFormProps~setPropertyValue("MasterFields", strArray)
strArray[1] = "c"
salesFormProps~setPropertyValue("DetailFields", strArray)
```

In order to be able to establish the connection between the two forms, a data field has to be defined for each form which creates finally the association. For the master form this is the customer ID, for the sub form it is the previously specified parameter “c”.

6.1.4 Create Components inside the Sub Form

In a next step the components inside the sub form needs to be created.

```
salesContainer = salesForm~XIndexContainer
call createControlAndShape "GridControl", 162, 40, 5, 50, salesContainer
```

Here a grid control is added to the container of the sub form. A grid control is a table-like component, which can comprise several columns of different form components. To add a new column to the grid control component the routine **`newGridColumn()`** is used. This routine is explained in the following cutouts.

```
/* query for the container to insert columns into */
xIndexContainer = arg_container~XIndexContainer
/* query for the factory for creating the column models */
```

```
xGridColumnFactory = arg_container~XGridColumnFactory
```

Firstly the *XIndexContainer* interface of the grid control, which is the first argument of the method, is retrieved. With the use of this interface new columns can be added to the grid control container. Secondly, we ask for the *XGridColumnFactory* interface, which can create new columns.

```
/* create a new column */
newColumn = xGridColumnFactory~createColumn(arg_factory)
xPropertySet= newColumn~XPropertySet
xPropertySet~setPropertyValue("DataField", arg_name)
xPropertySet~setPropertyValue("Label", arg_field)
/* insert the column */
xIndexContainer~insertByIndex(xIndexContainer~getCount, newColumn)
```

Thirdly, a new column is generated by defining a form component type in arg_factory. Afterwards the data field and the label of the column are set.

```
call newGridColumn gridControl, "ListBox", "name", "product name"
```

Now a new grid column can be created very easily via the `newGridColumn()` method as the previous cutout shows.

A special scenario is the adding of a list box. There the method call is followed by these lines of code:

```
productName = result --save the 'product name' column to set some more props

/* initialize the 'product name' ListBox to provide a choice of product names*/
productName~setPropertyValue( "ListSourceType", bsf.getConstant(-
        "com.sun.star.form.ListSourceType","SQL"))
productName~setPropertyValue("BoundColumn", box("sh","1"))

sListSource = "SELECT product.name, product.name FROM test.product"
strArray[1] = sListSource

productName~setPropertyValue( "ListSource", strArray )
```

The list box provides a choice of product names, if the user wants to add a new sale. The choice shall be defined through a “select statement”. For that reason the *ListSourceType* property has to be set to “SQL”. The property *BoundColumn* specifies which column of the result set should be used as the value of the component. In this example the data field where the value should be taken from and the data field

where the data should be stored to is the same. Therefore the same data field (“name”) is selected twice in the select statement.

6.1.5 Switch to the Live Mode

The creation of the form document is now completed. In OpenOffice.org there are two different view modes. It must be distinguished between the “design mode” and the “live mode”. The design mode is active during the design process of the form. In this mode you can insert form components, resize them and modify their properties. Since the form document has been built so far, the design mode is the current mode in this example. In order to allow the form to be connected to the database and additionally let the user interact with the form it has to be switched to the live mode. This is done by the method **`toggleFormDesignMode()`**.

```
URL = bsf.import('com.sun.star.util.URL')

aToggleURL = bsf.createArray(URL, 1)
aToggleURL[1] = URL~new
aToggleURL[1]~Complete = ".uno:SwitchControlDesignMode"
```

What this method does, is to dispatch the URL “`.uno:SwitchControlDesign-Mode`” into the current view. This simulates pressing the “Design Mode On/Off” button of the “Form Controls” toolbar. The first step is to create an instance of an new OpenOffice.org URL .

```
--need an URLTransformer
frameDesktop = .xMcf~createInstanceWithContext(-
        "com.sun.star.util.URLTransformer",.xContext)
xURLTransformer = frameDesktop~XURLTransformer
xURLTransformer~parseStrict(aToggleURL)
```

Afterwards this URL needs to be parsed by the *`URLTransformer`* service.

```
xController = .xComponent~XModel~getCurrentController~XController
--go get the dispatch provider of it's frame
xDispatchProvider = xController~getFrame~XDispatchProvider

xDispatch = xDispatchProvider~queryDispatch( aToggleURL[1],"", 0 )
xDispatch~dispatch( aToggleURL[1], .UNO~noProps )
```

Finally the *xDispatch* interface is needed to dispatch the previously created URL and consequently force the form to switch to the live mode. An instance of this interface is provided by the *DispatchProvider* service.

6.2 Add Form to the Database Document

After the form has been created it can be stored into the database document (i.e. the odb-File). Snippet 6.2 shows how the form can be added.

```
oDesktop = UNO.createDesktop() -- get the UNO Desktop service object
/* get the database document */
xComponentLoader = oDesktop~XDesktop~XComponentLoader
url = uno.ConvertToURL("c:\odbfiles\mysql1.odb")
xComponent = xComponentLoader~loadComponentFromURL(-
        url, "_blank", 0, .UNO~noProps)

/* get the form container */
xFormDocumentsSupplier = xComponent~XModel~XFormDocumentsSupplier
xHierarchicalNameAccess = xFormDocumentsSupplier~getFormDocuments

/* create a new DocumentDefinition containing the form */
props = bsf.createArray(.UNO~propertyValue,4)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "Name"
props[1]~Value = "test"
props[2] = .UNO~PropertyValue~new
props[2]~Name = "Parent"
props[2]~Value = xHierarchicalNameAccess
props[3] = .UNO~PropertyValue~new
props[3]~Name = "URL"
props[3]~Value = uno.ConvertToURL("c:\odbfiles\shop_form.odt")
props[4] = .UNO~PropertyValue~new
props[4]~Name = "DocumentTitle"
props[4]~Value = "test"

xDocMSF = xHierarchicalNameAccess~XMultiServiceFactory
oDBDocument = xDocMSF~createInstanceWithArguments(-
        "com.sun.star.sdb.DocumentDefinition", props)

/* add the DocumentDefinition to the database document */
xHierarchicalNameContainer = xHierarchicalNameAccess~XHierarchicalNameContainer
xHierarchicalNameContainer~insertByHierarchicalName("shop_form",oDBDocument)

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

Snippet 6.2 – Add Form to the Database Document

As a result the database document contains the form now as shown in Illustration 6.4.

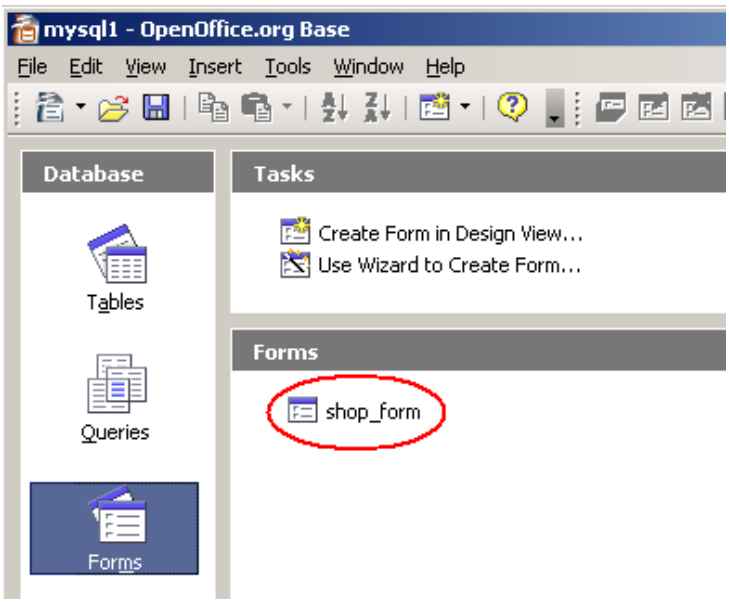

Illustration 6.4: Added Form.

The snippet explained in more detail:

```
oDesktop = UNO.createDesktop() -- get the UNO Desktop service object
/* get the database document */
xComponentLoader = oDesktop~XDesktop~XComponentLoader
url = uno.ConvertToURL("c:\odbfiles\mysql1.odb")
xComponent = xComponentLoader~loadComponentFromURL(-
        url, "_blank", 0, .UNO~noProps)
```

The first lines of code load the database document, which should be used to store the form inside. In this snippet the odb-File, which contains the computer shop database example from previous examples is taken (`mysql1.odb`).

```
xFormDocumentsSupplier = xComponent~XModel~XFormDocumentsSupplier
xHierarchicalNameAccess = xFormDocumentsSupplier~getFormDocuments
```

Afterwards the container, which holds the forms has to be retrieved. For that propose the *`XFormDocumentsSupplier`* interface posses the method `getFormDocuments`. At the same time there exists an interface called *`XReportDocumentsSupplier`*, which supports the method `getFormDocuments`. In this case the container with the reports inside is returned. If a report should be stored into the database document instead of an form, this report container has to be used equally.

```
/* create a new DocumentDefinition containing the form */
```

```
props = bsf.createArray(.UNO~propertyValue,4)
props[1] = .UNO~PropertyValue~new
props[1]~Name = "Name"
props[1]~Value = "test"
props[2] = .UNO~PropertyValue~new
props[2]~Name = "Parent"
props[2]~Value = xHierarchicalNameAccess
props[3] = .UNO~PropertyValue~new
props[3]~Name = "URL"
props[3]~Value = uno.ConvertToURL("c:\odbfiles\shop_form.odt")
props[4] = .UNO~PropertyValue~new
props[4]~Name = "DocumentTitle"
props[4]~Value = "test"

xDocMSF = xHierarchicalNameAccess~XMultiServiceFactory
oDBDocument = xDocMSF~createInstanceWithArguments(-
        "com.sun.star.sdb.DocumentDefinition", props)
```

The next step is to create a new DocumentDefinition with the form, that should be stored, inside. This service has to be created by the previously retrieved form container. In order to generate the DocumentDefinition an array of four properties needs to be passed. Important to mention here is the “Parent” parameter, which has to contain the form container. Furthermore the “URL” parameter comprises the URL of the form document, that should be added to the database document. Here the form document which was created in Snippet 6.1 on page 67 is used.

```
xHierarchicalNameContainer = xHierarchicalNameAccess~XHierarchicalNameContainer
xHierarchicalNameContainer~insertByHierarchicalName("shop_form",oDBDocument)
```

Finally the DocumentDefinition which represents the form document now, can be added to the form container.

7 Conclusion

The thesis shows how to automate the data source access of OpenOffice.org. Several new functions concerning the access to data sources has been introduced in OpenOffice.org 2.0, which facilitate the work with various databases. One new feature is the implementation of the Base application, which gives also, even not advanced users, the possibility to create connections to data sources within OpenOffice.org. This opportunity will enlarge the usage of OpenOffice.org.

The snippets provided, give an overview about the different possibilities to access data sources.

The Developers Guide of OpenOffice.org was obviously the first and a very good reference during the work on this thesis. Unfortunately, regarding to the database access some examples are outdated there. Additionally to several topics there could be more practical examples and a more detail explanation. The snippets of this thesis should help for further investigation on automation of the database access module.

Although several automation topics are provided in this paper, there is much more potential behind the automation of OpenOffice.org regarding to data sources.

8 List of Snippets

Snippet Name	*File Name*	*Page*
Snippet 4.1	s4.1_print_registered_data_sources.rex	30
Snippet 4.2	s4.2_addingDatasource_adabas_d.rex	33
Snippet 4.3	s4.3_addingDatasources_mysql.rex	37
Snippet 4.4	s4.4_connection_through_a_datasource.rex	40
Snippet 4.5	s4.5_connection_using_DriverManager.rex	43
Snippet 4.6	s4.6_md_statement.rex	46
Snippet 4.7	s4.7_md_rowset.rex	51
Snippet 5.1	s5.1_create_predefined_query.rex	55
Snippet 5.2	s5.2_execute_predefined_query.rex	58
Snippet 6.1	s6.1_create_form.rex	67
Snippet 6.2	s6.2_add_form.rex	77

9 List of Illustrations

Illustration Index

Illustration 2.1: Customer Table....8

Illustration 2.2: Connection of Tables Customer and Sales....8

Illustration 2.3: ER-Diagram of a Computer Shop....9

Illustration 2.4: Table Format of the Computer Shop ER-Diagram.... 10

Illustration 3.1: Configuring OpenOffice.org Applications from UNO components. [Flat05] p.5....23

Illustration 3.2: DatabaseContext's interfaces....25

Illustration 3.3: From ooRexx to OpenOffice.org. [Prem06] p.24.... 27

Illustration 4.1: Registered Databases....30

Illustration 4.2: Output of Snippet 4.2....33

Illustration 4.3: Output of Snippet 4.4....40

Illustration 4.4: Database Login....41

Illustration 4.5: Output of Snippet 4.6....47

Illustration 4.6: Output of Snippet 4.7....51

Illustration 4.7: Base Tables....52

Illustration 5.1: Output of Snippet 5.1....55

Illustration 5.2: Output of Snippet 5.2....59

Illustration 6.1: Master Form - Sub Form Relation....61

Illustration 6.2: Form Navigation Toolbar.....61

Illustration 6.3: Output of Snippet 6.1.....67

Illustration 6.4: Added Form.....78

10 Bibliography

[Aham05] Andreas Ahammer: OpenOffice.org Automation: Object Model, Scripting Languages, “Nutshell”-Examples, Bachelor Course Paper, 2005
http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2005/200511_OOo-Ahammer/200511_OOoAutomation.pdf
retrieved on 2007-02-01

[Apac01] Apache Software Foundation: The Apache Jakarta Project – Bean Scripting Framework
http://jakarta.apache.org/bsf/index.html
retrieved on 2007-02-01

[Apac02] Apache Software Foundation: The Apache Jakarta Project – BSF Architectural Overview
http://jakarta.apache.org/bsf/manual.html
retrieved on 2007-02-01

[ApiO03-1] Sun Microsystems, Inc.: OpenOffice.org API – IDL Reference, 2003
http://api.openoffice.org/docs/common/ref/com/sun/star/sdb/DatabaseContext.html
retrieved on 2007-02-01

[ApiO03-2] Sun Microsystems, Inc.: OpenOffice.org API – IDL Reference, 2003
http://api.openoffice.org/docs/common/ref/com/sun/star/sdbc/ResultSet.html
retrieved on 2007-02-01

[ApiO03-3] Sun Microsystems, Inc.: OpenOffice.org API – IDL Reference, 2003
http://api.openoffice.org/docs/common/ref/com/sun/star/sdb/RowSet.html
retrieved on 2007-02-01

[Burg06] Martin Burger: OpenOffice.org Automation: OpenOffice.org Automatisation with Object Rexx, Bachelor Course Paper, 2006
http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2006/200605_Burger/Bakk_Arbeit_Burger20060519.pdf
retrieved on 2007-02-01

[Deve05] Sun Microsystems, Inc.: OpenOffice.org 2.0 – Developer's Guide, 2005
http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf
retrieved on 2007-02-01

[Flat05] Rony G. Flatscher: AUTOMATING OPENOFFICE.ORG WITH OOREXX: ARCHITECTURE, GLUING TO REXX USING BSF4REXX
, 2005, International Rexx Symposium, Austin, Texas, U.S.A.
http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf#search=%22AUTOMATING%20OPENOFFICE.ORG%20WITH%20OOREXX%3A%22
retrieved on 2007-02-01

[Flat06-1] Rony G. Flatscher: The Vienna Version of BSF4Rexx, 2006, International Rexx Symposium, Austin, Texas, U.S.A.
http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf
retrieved on 2007-02-01

[Flat06] Rony G. Flatscher: Resurrecting REXX, Introducing ObjectRexx, 2006, Vienna University of Economics and Business Administration, Austria
http://prog.vub.ac.be/%7Ewdmeuter/RDL06/Flatscher.pdf
retrieved on 2007-02-01

[Geis05] Frank Geisler: Datanbanken – Grundlagen und Design, 2005, mitp-Verlag/Bonn, Germany

[HaNe05] Hansen / Neumann: Wirtschaftsinformatik 1 – Grundlagen und Anwendungen 9.Auflage, 2005, Lucius&Lucius, Stuttgart

[Hinz06] Michael Hinz: OpenOffice.org Calc Automation Using ooRexx, Bachelor Course Paper, 2006
http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2006/200607_Hinz/20060712_OOo_calc_automation.pdf
retrieved on 2007-02-01

[OASIS06] OASIS – About OASIS
http://www.oasis-open.org/who/
retrieved on 2007-02-01

[ooRx06-1] Rexx Language Association: Open Object Rexx - Programming Guide, 2006
http://www.oorexx.org/rexxpg/book1.htm
retrieved on 2007-02-01

[ooRx06] Rexx Language Association – Open Object Rexx, 2006
http://www.oorexx.org/
retrieved on 2007-02-01

[Open06-1] Sun Microsystems, Inc.: OpenOffice.org – About Us, 2006
http://about.openoffice.org/index.html
retrieved on 2007-02-01

[Open06-2] Sun Microsystems, Inc.: OpenOffice.org – Product Description
http://www.openoffice.org/product/
retrieved on 2007-02-01

[Open06] OpenOffice.org – Writer Guide Second Edition, 2006
http://documentation.openoffice.org/manuals/oooauthors2/0200WG-WriterGuide.pdf
retrieved on 2007-02-01

[OpenSR] Sun Microsystems, Inc.: OpenOffice.org – System Requirements for OpenOffice.org
http://www.openoffice.org/dev_docs/source/sys_reqs.html
retrieved on 2007-02-01

[Prem06] Matthias Prem: ooRexx Snippets for OpenOffice.org Writer, Bachelor Course Paper, 2006
http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2006/200607_Prem/20060724_ooRexxSnippetsOOoWriter_2.1_odt.pdf
retrieved on 2007-02-01

[RexxLA06] Rexx Language Association - The Rexx Language Association, 2006
http://www.rexxla.org/
retrieved on 2007-02-01

[SKSu02] Abraham Silberschatz, Henry F. Korth, S. Sudarshan: Database System Concepts - 4th Edition, 2002, McGrawl-Hill, New York

[StarO1] Sun Microsystems, Inc.: StarOffice 8 – Tech FAQs
http://www.sun.com/software/star/staroffice/faqs/technical.jsp
retrieved on 2007-02-01

[Wiki06-1] Wikipedia – REXX, 2006
http://en.wikipedia.org/wiki/Rexx
retrieved on 2007-02-01