WIRTSCHAFTSUNIVERSITÄT WIEN

BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:
Automatisierung von OpenOffice.org mit Hilfe der Skriptsprache ooRexx anhand ausgewählter Kurzbeispiele von Andrew Pitonyak.

Englischer Titel der Bakkalaureatsarbeit:
OpenOffice.org Automation Using ooRexx Scripting Language by means of Selected Nutshell Examples by Andrew Pitonyak.

Verfasser:	Christoph Waglechner
Matrikelnummer:	0204938
Studienrichtung:	Wirtschaftsinformatik
Kurs:	0780 Vertiefungskurs VI – Electronic Commerce
Textsprache	Englisch
Betreuer:	ao. Univ.-Prof. Dr. Rony G. Flatscher

Ich versichere,

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

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

dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

_____ _____

Datum Unterschrift

Abstract

OpenOffice.org, an open-source office suite, implements a flexible scripting framework, which allows macro programming using various languages, as long as they can interact with UNO (Universal Network Object) architecture used to address OpenOffice.org objects. For the easy-to-use open-source scripting language ooRexx (Open Object Rexx), this connection is supplied by the freely available BSF4REXX framework.

Within these settings, the thesis provides additional examples on how to script OpenOffice.org using ooRexx, which are mainly derived from Andrew Pitonyak's OpenOffice.org macro guide. For Writer, selected examples on how to travel through the text using cursors, insert text fields and form letters as well as OpenOffice.org Math integration are covered. The Calc chapter demonstrates basic cell range functionalities like cell queries as well as formula operations, which include array formulas and multiple operations. Furthermore, a generic section is provided which shows how the suite's modules deal with graphics in general.

All examples in this document only contain a minimal connection creation procedure allowing invocation from within OpenOffice.org. To replace it with a more sophisticated variant, copy the listing from section 2.1.1 into the files.

Key Words
OpenOffice.org, ooRexx, Pitonyak Nutshells, Office Automation, BSF4REXX, UNO, Open Object Rexx, Script, Macro

Contents

Listings

List of Figures

1. Introduction

OpenOffice.org, “the leading open-source office software suite” [Opend], was originally developed by a German company, “Star Division”. It was portable to MacOS, Unix, Operating System/2 and Windows and built to cooperate with Microsoft Office in terms of file formats, as import- and export-filters for Microsoft formats were provided, GUI (Graphical User Interface), and even integrated scripting language [Flat05a].

After Sun bought the product, it both extended the office suite to be programmable from Java and released it to open source [Flat05a], currently OpenOffice.org is published under terms of the LPGLv3 (GNU Lesser General Public License v3, see [Free07]) [Openc, /license.html].

By version 3.0.1, OpenOffice.org includes

- Writer as a word processor,
- Calc as a spreadsheet application,
- Impress for presentations,
- Base providing a database management program,
- Draw, which is a vector graphics application and
- Math, a tool for creating and manipulating formulas.

Apart from the integrated scripting language called OpenOffice.org Basic, which is structurally similar to Microsoft’s Visual Basic, starting with OpenOffice 2.0, the office suite also offers a Java based scripting framework [Flat05a]. It is openly designed, thus support for new languages can be added [Sun07, p. 1095] to the by default available Java, JavaScript, BeanShell and Basic [Sun07, p. 1105], but also to all languages having a so-called UNO bridge (Universal Object Network bridge), e.g. Python, C++ or Java [Opena, /scripting]. For the easy-to-learn scripting language ooRexx [Rexx] there exists a bridge to Java, BSF4REXX [Flat08a]. Java offers a UNO bridge — resultingly, ooRexx can be used to script OpenOffice.org [Flat06].

1.1. OpenOffice.org – Architecture

OpenOffice.org is designed as a client-/server-architecture. Although normally both components are installed on one system, they can also reside on different machines and operating systems. It communicates via TCP/IP or named pipes using the CORBA-like urp protocol [Flat06], which acts as a kind of a bridge between objects written in different languages [Sun07]. CORBA, the Common Object Request Broker Architecture, is a standard which aims at a normalization of method calls between applications. It implements IDL (interface definition language), which specifies interfaces of objects to the outside, which are then mapped in CORBA to the specific implementations in different languages [Wiki, /CORBA]. The component model in OpenOffice.org, UNO, uses IDL to describe its components, which can be shared through program modules [Flat05a]. This re-use allows the macro programmer to use the same concepts throughout all program modules provided by OpenOffice.org [Flat05a]. UNO thus allows for providing objects accross programming and platform boundaries [Sun07].

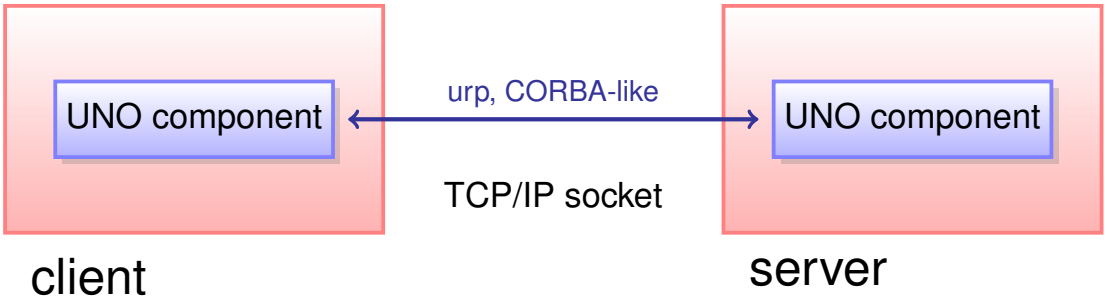


Figure 1.1.: OpenOffice.org client/server architecture (taken from [Flat05a]).

Components in targets are created with a factory concept, thus an abstraction of constructors. The factories in UNO are called “service managers”, which hold all registered components that can be created by name indifferent of the component’s implementation language. Versions newer than 1.0 have the “component context” object as being central in a UNO application, which it is passed to during initialization. It allows the retrieval of a service manager [Sun07].

An “interface” of an object gives a certain aspect or functionality of an object by publishing a set of operations, mainly belonging semantically together, without unveiling the internals. The interfaces’ names in UNO start with a capital “X”. One interface thus gives only one view on an object, which might not be sufficient. UNO therefore provides “multiple-inheritance interfaces” as well as “services”, together both are able to completely specify an object. The first combines various aspects of an object represented by single-inheritance interfaces, in the second step a service description must be provided if the object is available at the global component context [Sun07]. Therefore, a service is the definition the UNO object is created upon, apart from the

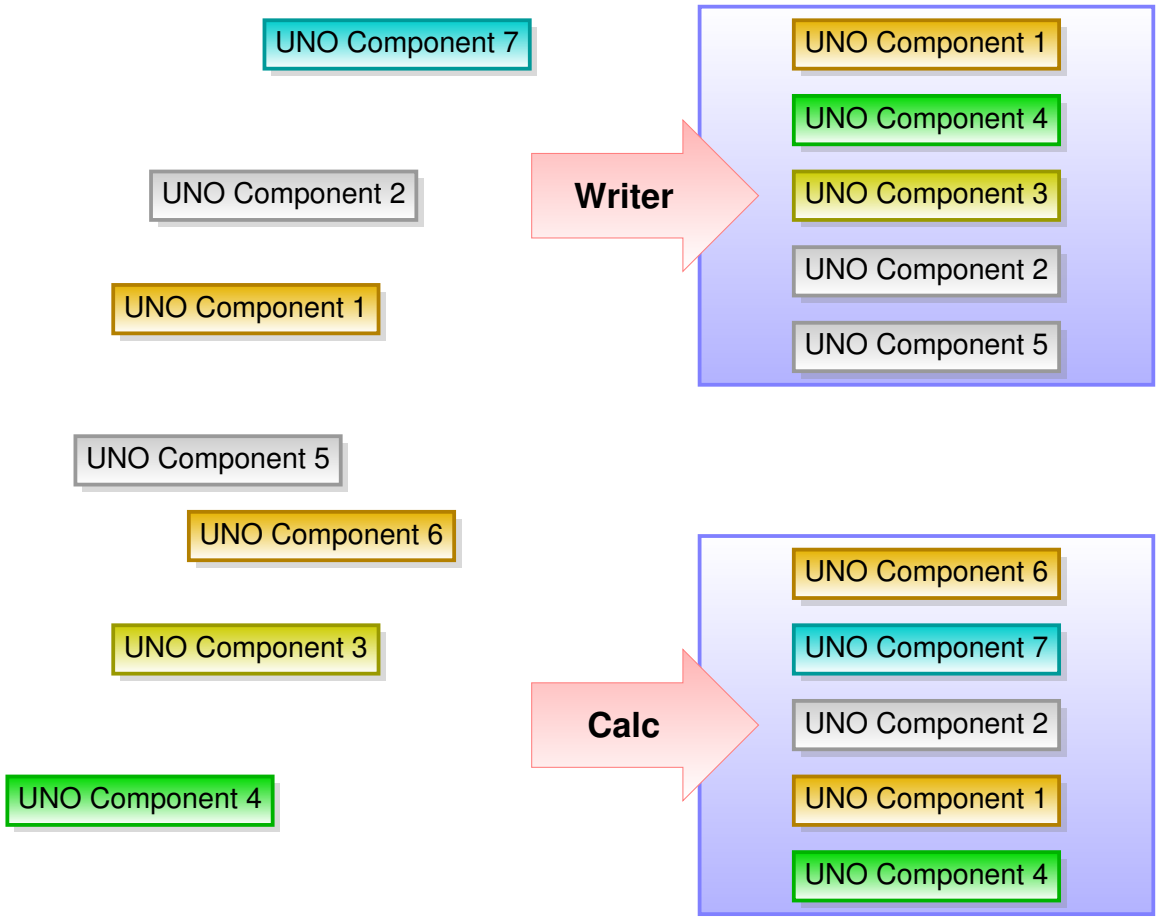


Figure 1.2.: UNO components can be reused in various OpenOffice.org modules (taken from [Flat]).

interfaces a service might also consist of UNO structures, which are grouped standard types like integers or strings combined [Pito04].

The service manager is able to instantiate services using its `XMultiComponentFactory` and `XMultiServiceFactory` interfaces. The second passes all calls to the first, which is advised to be used. The `XMultiComponentFactory` can supplement necessary parameter from its own `XComponentContext` interface [Sun07]. If a service's method is needed, at first the interface has to be queried resulting in the return of another object additionally implementing the requested methods, which afterwards can be called. The process is initiated with the `queryInterface()` method implemented in the `com.sun.star.uno.XInterface`, which is the interface all UNO interfaces inherit from. According to Flatscher, this procedure ensures a locally minimal interface driving a remote object, it gives a better overview as functionally and semantically similar methods are grouped together, and over the time the programmer gets a good knowledge on the organization of OpenOffice.org [Flat05a].

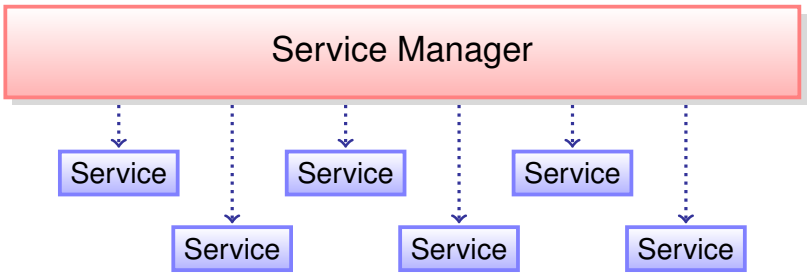


Figure 1.3.: The service manager creates services [Sun07, p. 38].

1.2. ooRexx and BSF4REXX

REXX is an interpreted programming language originally developed by IBM, with the first specification fixed in 1979. In 2004, IBM announced an open-source release of the object-oriented implementation Object REXX. Firstly published in 2005, it is called ooRexx (Open Object Rexx) [Wiki, REXX, Open_Object_Rexx]. ooRexx is available on MacOS X, AIX, Solaris, Linux and Windows. By June 2009, the latest stable release is version 3.2.0-1, version 4.0 is in the beta testing phase [Rexx]. The ooRexx-Brochure produced by the RexxLA, the Rexx Language Association managing the project, lists

- its open-source availability,
- its natural language syntax, which is said to make programming easy,
- its implicit data typing to reduce complexibility,
- its cross-platform availability which keeps development costs low,
- its decimal arithmetic and
- its object-orientation

as ooRexx' biggest advantages [Rexx07]. These features make ooRexx a prospective partner for scripting OpenOffice.org.

Nonetheless, there does not exist a native ooRexx UNO bridge as for Java, C++, Tcl or Python [Wiki, Universal_Network_Objects], making a direct connection impossible. Starting with a proof-of-concept study by Kalender within a seminar 2000/2001 at the University of Essen, Flatscher has been developing "BSF4REXX" [Flat01]. This framework adds Rexx-support to the BSF, the Bean Scripting Framework. BSF is an open-source framework maintained by the Apache Jakarta Project [Apac], which allows Java to execute scripting languages by providing an API. It also supports scripting languages to access Java objects via an object registry [Apac].

ooRexx is therefore able to interact with Java via BSF4REXX, which additionally is freely available from [Flat08a].

The BSF4REXX package does not only include the Rexx-to-Java-bridge, it also supplies the language script provider necessary to execute ooRexx macros from within OpenOffice.org as well as UNO.cls, which drastically enhances the generic OpenOffice.org scripting support. As an example, UNO.cls provides easy-to-use methods to connect to OpenOffice.org (`uno.connect()`) or to retrieve an **`XDesktop`** (`uno.createDesktop()`) or the script context if ooRexx is invoked from within OpenOffice.org (`getScriptContext()`). Additionally, the package allows introspection and reflection, as e.g. provided by a routine listing all interfaces object `o` implements, `uno.getInterfaceNamesViaReflection(o)` [Flat06].

1.3. Environment and Setup

The examples presented in chapters 3–5 are generated and tested using the following environment:

- Microsoft Windows XP Professional Service Pack 3
- OpenOffice.org 3.0.1
- Java SE Runtime Environment 1.6.0_07-b06
- ooRexx 3.2.0
- BSF4REXX Vienna Version 2008-09-12; including its OpenOffice.org support and its ScriptProviderForooRexx

A guide on how to install the program packages and on how to glue ooRexx to OpenOffice.org can be found within the help files provided with BSF4REXX [Flat08a] as well as within Frysak [Frys08], for the test installation above the automated installation scripts contained in the BSF4REXX package worked out of the box. The OpenOffice.org Java support has to be enabled, which can be done in version 3.0.1 via Extras–Options–OpenOffice.org–Java, where one of the currently installed Java runtimes can be selected. If Ubuntu 8.04 is used, the standard version of OpenOffice.org 2.4 nonetheless does not incorporate Java support, this version has to be downloaded and installed separately (package openoffice.org-java-common) [ubun].

2. A Brief Roundup on Fundamental ooRexx Scripts Automating OpenOffice.org

The automation of OpenOffice.org using ooRexx and the BSF4REXX framework has already been covered in literature, e.g. [Frys08, Gmei08, Hinz06, Aham05, Burg06, Scho07, Scho09]. Many scripts of these authors can be used as an examplary approach to the problems tackled in this work, additionally many basic items, e.g. the connecting procedure from ooRexx to OpenOffice.org, are needed in all nutshell examples. We therefore shortly introduce the preliminary work already developed, with a strong focus on the part related to the new examples presented in chapters 3–5.

2.1. General OpenOffice.org Environment

The general OpenOffice.org environment scripts deal with problems common for all applications, e.g. on how to invoke a macro.

2.1.1. Macro execution and invocation in OpenOffice.org

The most fundamental aspect of an OpenOffice.org automation is creating a way to execute a macro. In principle, there are two possibilities to succeed, the first calling a macro externally, thus from the system command line, the second from within a library of OpenOffice.org macros. Frysak gives a good overview on how to manage these methods [Frys08]. It is important to mention that after a successful connection to OpenOffice.org has been established and the basic UNO proxy objects have been retrieved, both variants result in the same code. Ideally, a macro therefore is indifferent to the environment it is called from. One practicable solution was developed by Frysak [Frys08].

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
xsc = uno.getScriptContext()
if (xsc <> .nil) then
do
-- invoked by OOo as a macro
xcc = xsc~getComponentContext
xdt = xsc~getDesktop
xdoc = xsc~getDocument
end
else
do
-- called from outside of OOo, create a connection
xcc = UNO.connect()
-- create a desktop service and its interface
service = "com.sun.star.frame.Desktop"
s_Desktop = xcc~getServiceManager~XMultiServiceFactory~createInstance(service)
xdt = s_Desktop~XDesktop
xdoc = xdt~getCurrentComponent()
end
::requires UNO.cls
```

Listing 2.1: litex07_Connection.rex. Externally and internally called macros [Frys08].

In listing 2.1, Frysak first tries to obtain the script context using a public routine specifically provided by the UNO.cls, `uno.getScriptContext()`. It returns a UNO proxy allowing the retrieval of the `XModel`, `XDesktop` and `XComponentContext` objects. If not called from within OpenOffice.org, thus e.g. from the command line, the routine returns the `.nil` object [Flat08b].

This information is vital to decide on object obtaining. For an internal call, they now can simply be gotten (listing 2.1, line 5). The second part deals with external execution, where first an `XComponentContext` object is obtained by issuing the `uno.connect()` method of the UNO.cls [Flat08b]. This `XComponentContext` is used to get the service manager, which can create a desktop service via `createInstance()`. The desktop service finally implements `XDesktop` and offers a method to retrieve the current document.

To sum up, Frysak obtains at any case the `XModel`, `XDesktop` and `XComponentContext` UNO proxy objects which can later be used as a starting point for scripting commands [Frys08]. With OpenOffice 3.0, the `getInvocationContext()` method of the `XInvocationContext` interface allows to access the context it was invoked from, if not the document itself [Openb, script/provider/XScriptContext], [Flat08a, Default Example].

2.2. OpenOffice.org Writer

For the included Writer module, many scripts using ooRexx have already been published. The following section concentrates on cursor management, graphics and text fields, as these are the main topics addressed in chapter 3.

2.2.1. Cursor Creation and Usage

Apart from some calls affecting the whole document, e.g. saving or printing, in most cases users only want to modify parts of a document, for which OpenOffice.org provides so-called *cursors*. There exist two basic types of cursors in Writer documents: On the one hand a text cursor provided by the `XTextCursor` interface, which extends `XTextRange` by methods of position modifying [Openb, XTextCursor]. On the other hand, the view cursor (`XViewCursor`) refers to the visible cursor blinking in the document. OpenOffice.org compasses different view cursors, e.g. the `XTextViewCursor` which is derived from the `XTextCursor` [Pito04, p. 282]. As the view cursor knows the layout and how text is displayed, it can be used to go through the text by means of lines or pages [Pito04].

Text Cursors

The basic ooRexx usage of text cursors within a text or draw document has extensively been documented, e.g. in [Aham05] who implemented writer document statistics with the help of a cursor traversing the text. He also inserts tables and text in Writer, Prem special characters [Prem06]. Scholz showed the use of a text cursor in Draw documents, where it is necessary to modify the text of shapes; Frysak used the text cursor's `XDocumentInsertable` interface to import files into the current Writer document [Frys08]. To sum up, text cursors can be used for [Sun07]

- traversing through words, sentences and paragraphs,
- inserting text, special characters or other items,
- deleting or modifying text,
- selecting a range within a document using the `XTextRange` interface,
- changing text properties in a selected range via the `XPropertySet`
- modifying text and its properties in shapes
- importing files with the help of `XDocumentInsertable` interface

In listing 2.2, lines 6–18 illustrate the retrieval and use of a text cursor and its focus on the inner text structure consisting of words, sentences and paragraphs [Aham05, Frys08, Prem06].

View Cursors

In contrast to the well-described text cursor, only a few examples show the application of a view cursor. Ahammer as well as Burger demonstrate the basics and use it for page counting and page number setting [Aham05, Burg06], Burger also shifts the displayed page via **XScreenCursor** and Prem changes the font of a selected range [Prem06]. Another usage is shown by Frysak, who determines the current position within the text [Frys08].

In contrast to the text cursor, which has to be created via the **XText**, the only view cursor in a document is available via the current controller which can be obtained from the document using **getCurrentController**() and **getViewCursor**() of **XTextViewCursorSupplier**. It is responsible for elements which are dependent on the current formatting like lines or pages. Lines 20–29 in listing 2.2 examplify the page-dependent and line-dependent usage of the viewcursor.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/**Text Cursor**/
/*basic text cursor methods described in Ahammer2005*/
xText = xdoc~XTextDocument~getText()
oTextCursor = xText~createTextCursor()
s="*** Text Cursor Usage Examples ***" .ENDOFLINE
/*send to start*/
oTextCursor~gotoStart(.false)
oTextCursor~goRight(0,.false)
oTextCursor~XWordCursor~gotoNextWord(.true)
s =s||"Word:" oTextCursor~XTextRange~getString() .ENDOFLINE
oTextCursor~collapseToStart()
oTextCursor~XSentenceCursor~gotoNextSentence(.true)
s = s||"Sentence:" oTextCursor~XTextRange~getString() .ENDOFLINE

/**View Cursor**/
s = s||"*** View Cursor Usage Examples ***" .ENDOFLINE
/*basic view cursor methods described in Ahammer2005, Burger2006*/
oViewCursor = xdoc~getCurrentController()~XTextViewCursorSupplier~getViewCursor()
iCurPage = oViewCursor~XPageCursor~getPage()
s = s||"Current Page:" iCurPage .ENDOFLINE
if oViewCursor~XLineCursor~isAtStartOfLine() then
s = s||"View Cursor at start of line."
else
s = s||"View Cursor is not at start of line."

.bsf.dialog~messageBox(s)

::requires UNO.cls
```

Listing 2.2: litex02_Cursor.rex. Examplifies the usage of various cursors [Aham05, Burg06].

2.2.2. Object and Text Insertion

In Writer, objects and texts are normally inserted with the help of the cursors already discussed in section 2.2.1. Ahammer described how to use them to import text from another OpenOffice.org document using the **XDocumentInsertable** interface implemented by the text cursor and its **insertDocumentFromURL()** method expecting a valid URL [Aham05]. Objects like tables or table of contents [Prem06] can be inserted into a document at the cursor's position using the **insertTextContent()** method supplied by the **XText** interface. This interface can be accessed for the whole text via the **XTextDocument** interface and its **getText()** method, which is also described in [Aham05], but also only for the text range of a cursor, as its **XTextRange** interface also implements the **getText()** method. Subsequently, **insertTextContent()** is used for inserting text fields (e.g. [Frys08]) as well as images (e.g. [Gmei08] and sections 2.2.3 and 2.2.4). Its three parameters are the cursor at which the second parameter, the content, should be inserted and a flag, which is **.true** if the cursor's currently selected text is to be replaced.

Normal text can be inserted via **insertString()** of the **XSimpleText** interface [Aham05] (see listing 2.3). Three parameters are needed, the text cursor where the string is to be inserted, the string itself and a flag indicating whether the text selected by the text cursor should be replaced [Openb, text/XSimpleText]. The same interface supplies **insertControlCharacter()**, already used by Ahammer [Aham05], which is similarly used and inserts various control characters defined in the constants group `com.sun.star.text.ControlCharacter`, e.g. `"PARAGRAPH_BREAK"` or `"HARD_SPACE"` [Openb, text/ControlCharacter].

To illustrate the contrary operation, the whole text can be retrieved using **getString()** of the cursor's **XTextRange** interface (first ooRexx use in [SpRG06]). In contrast to ooBasic, where strings are limited to 64KB [Pito04], in ooRexx this also works for large texts.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*create a cursor and set to the start*/
xText = xdoc~XTextDocument~getText()
oCursor = xText~XSimpleText~createTextCursor()
oCursor~collapseToStart()

/*insert a string at the beginning*/
xText~XSimpleText~insertString(oCursor,"This is a teststring.",.false)
/*insert a control character, at the end of the former insertion*/
xText~insertControlCharacter(oCursor,bsf.getStaticValue("com.sun.star.text.ControlCharacter","
    PARAGRAPH_BREAK"),.false)
/*insert a second string, at the end of the former insertion*/
xText~XSimpleText~insertString(oCursor,"This is a second teststring.",.false)

/*get the whole visible text*/
oCursor~gotoStart(.false)
oCursor~collapseToStart()
oCursor~gotoEnd(.true)
sText = oCursor~XTextRange~getString()
```

```
.bsf.dialog~messageBox("Current Text length:" sText~length)

/*alternative insertion - Prem2006*/
oCursor~collapseToEnd()
oCursor~XTextRange~setString(" Another insertion method.")

::requires UNO.cls
```

Listing 2.3: litex03_Insertion.rex. A minimal script demonstrating text insertion [Aham05].

2.2.3. Text Fields

Text fields are special insertions which add a second information level to the document. Typically, although their appearance seamlessly integrates with the surrounding text, their content is taken from another source, e.g. a database or the result of a field command. Prominent examples of text fields are the current page number, the whole pages number, the current date or a cross-reference to another area of the text [Sun07].

The basic handling of text fields and their insertion into the text has already been presented by Ahammer [Aham05] for Writer documents, further applicable examples can be found in Burger [Burg06], who also used the "PageNumber" field, Frysak [Frys08] (additionally use of "Annotation" and "URL") and Scholz [Scho09]. Most applications include the "DateTime" text field, which represents the date and time at creation time. Scholz also used the "URL" text field coding a URL and various document statistics text fields, e.g. current page number [Scho09].

A minimal example first has to create the text field via the **`XMultiServiceFactory`** of the model, second the field's properties are modified using the **`XPropertySet`** of the obtained field, and last it has to be inserted into the text using the **`insertTextContent`**`()` method of the document text returned by an **`XTextDocument`** method, **`getText`**`()`.

```
xsc = uno.getScriptContext()
xcc = xsc~getComponentContext
xdt = xsc~getDesktop
xdoc = xsc~getDocument

/* create the TextObject and the TextCursor */
xTextDocument = xdoc~XTextDocument
xText = XTextDocument~getText()
xTextCursor = xText~createTextCursor()

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

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

```
22
   ::requires UNO.cls -- get UNO support
```

Listing 2.4: litex06_TextFields.rex. Inserts a “DateTime” text field and modifies a property [Aham05].

In listing 2.4, the `"IsDate"` property is set to `.true`, which causes the field to contain the current date. All text field services implement the `com.sun.star.text.TextField` service, [Sun07, p. 589] gives an overview.

2.2.4. Graphics in Writer

To insert a graphic, two services can be used: `com.sun.star.text.TextGraphicObject` and `com.sun.star.drawing.GraphicObjectShape`. Both are shape objects, they act independently of ordinary text flow, they can be wrapped by the text, or lie behind or in front of it. Via anchoring, it is possible to attach a shape to a text [Sun07, p. 602]. `TextGraphicObject` is derived from `com.sun.star.text.BaseFrame`, which is responsible for embedding various objects and files like Draw or Calc documents, no additional interface is provided. Nonetheless, some graphic management properties are added. The `GraphicObjectShape` on the other hand is an OpenOffice.org's drawing shape, which is derived from `com.sun.star.drawing.Shape`. As it supports the **`XTextContent`** interface, it can be added to an **`XText`** [Sun07, p. 602–604], a concept which is in principle availabe in all OpenOffice.org modules, but in Writer no 3D objects are supported [Sun07, p. 610].

Resultingly, as far as practical applications are concerned, both types fulfil basic graphic requirements. While `TextGraphicObject` supplies a graphics dialog, `GraphicObjectShape` provides the shape dialog when right-clicked with the mouse. The already existing examples inserting graphics into text also use both types, the `TextGraphicObject` is used by Gmeiner [Gmei08], while Burger instantiates `GraphicObjectShape` for his Writer shapes demonstrations [Burg06]. For Calc and other OpenOffice.org modules (e.g. [Hinz06, Frys08, Gund07]), the `GraphicObjectShape` is used. If the user doesn't feel save with manually specifying paths to filenames, listing A.6 in section A.2 provides a graphic file selection dialog for this example.

```
   /*standard header*/
2  xsc = uno.getScriptContext()
   xdt = xsc~getDesktop()
4  xdoc = xsc~getDocument()

6  /* create the TextObject and the TextCursor */
   xTextDocument = xdoc~XTextDocument
8  xText = XTextDocument~getText()
   xTextCursor = xText~createTextCursor()
10 xDMsf = xTextDocument~XMultiServiceFactory

12 sGraphicURL = "c:\temp\graphic.png"
```

```
14 sGraphicURL = uno.convertToURL(sGraphicURL)
   size = .bsf~new("com.sun.star.awt.Size")
16 size~Height = 1160
   size~Width = 4000
18
   /****GraphicObjectShape****/
20 /*Burger 2006*/
   oGraph = xDMsf~createInstance("com.sun.star.drawing.GraphicObjectShape")
22 xGraph = oGraph~XShape
   xGraph~setSize(size)
24 xPropertySet=xGraph~XPropertySet
   xPropertySet~setPropertyValue("GraphicURL", sGraphicURL)
26 xText~insertTextContent(xText~getEnd,oGraph~XTextContent, .false)

28 /****TextGraphicObject****/
   /*Gmeiner 2008*/
30 oTextGraphic = xDMsf~createInstance("com.sun.star.text.TextGraphicObject")
   xGraphicProperties = oTextGraphic~XPropertySet
32 xGraphicProperties~setPropertyValue("GraphicURL", sGraphicURL)
   xGraphicProperties~setPropertyValue("AnchorType",bsf.getConstant("
       com.sun.star.text.TextContentAnchorType", "AS_CHARACTER"))
34 xGraphicProperties~setPropertyValue("Size",size)
   xText~insertTextContent(xText~getEnd, oTextGraphic~XTextContent, .false)
36
   ::requires UNO.cls
```

Listing 2.5: litex01_GraphicInsertion.rex. Two possibilities of image insertion [Burg06, Gmei08].

Listing 2.5 examplifies both variants with the same `sGraphicURL` and the same `size`. In both cases, the model's multi service factory has to instantiate the image service, the insertion into the document is equal, too, as well as the setting of the graphic's file source. `GraphicObjectShape` offers its own functions for modifying its size and position via the `setSize()` (line 23) and `setPosition()` methods as it implements the **`XShape`** interface, while for the `TextGraphicObject` all properties are set via the **`XPropertySet`** (line 34).

Apart from basic graphic insertion examples, only a few scripts demonstrate the graphic capabilities of Writer. Frysak implemented a macro which exports all graphics of a text document via the **`XTextGraphicObjectsSupplier`** interface of the document model, which only method `getTextGraphicObjects()` returns all `TextGraphicObject` graphics [Sun07, p. 603].

2.3. OpenOffice.org Calc

The brief roundup on Calc especially focuses on cells and cell range addressing, as these most basic operations are needed in nearly all scripts.

2.3.1. Cell Addressing

The most principle task in OpenOffice.org Calc is gaining access to the cell content. Ahammer demonstrated the usage of the special ooRexx-UNO.cls function `uno.setCell`, which easily

allows to set the cell via four parameters, the first indicating the sheet, an **XSpreadsheet** interface, the following two referencing cell and row starting with 0, and the fourth being the content [Aham05, Flat08b].

The sheet can be retrieved via the **XSpreadsheetView** interface of the document's controller, which **getActiveSheet()** method returns the currently active sheet. **uno.setCell()** internally uses **setFormula()** of the **XCell** interface, which is returned by a **getCellByPosition()** call on the sheet. The **setFormula()** method is language-independent [Flat08a, UNO.cls], thus the dot is always used as separator for floats and all built-in functions have to be specified by their English name. Setting cell values without calling **uno.setCell()** is demonstrated in [SpRG06]. Additionally, UNO.cls offers **uno.getCell** method which returns a cell values.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

oController = xdoc~XModel~getCurrentController()
xSheet = oController~XSpreadsheetView~getActiveSheet()

/*UNO short call*/
call uno.setCell xSheet, 0, 0, "=SIN(0.75)"
/*equivalent long version*/
xCell = xSheet~getCellByPosition(0,1)
xCell~setFormula("=SIN(0.75)")

/*alternative UNO call to set cell*/
CALL uno.setCell xSheet,"B1","=SIN(0.9)"
/*equivalent long version*/
xCellRange = xSheet~getCellRangeByName("B2")
xCell = xCellRange~getCellByPosition(0,0)
xCell~setFormula("=SIN(0.9)")

::requires UNO.cls
```

Listing 2.6: litex04_CalcCells. Shows how to access Calc cells [Aham05].

2.3.2. Cell Ranges and Selections

Cell ranges are a rectangular range of cells [Sun07]. Their handling is as important as that of single cells. Ahammer demonstrated their access via **getCellRangeByName()** (see section 2.3.1). Their address representation of type `com.sun.star.table.CellRangeAddress` is a structure definitely specifying a cell range within one Calc document, as it has a `"Sheet"` field as well as start and end columns and row fields [Openb, table/CellRangeAddress].

Gmeiner additionally obtained the cell range in use via the **XUsedAreaCursor** interface of a spreadsheet cursor created with the **createCursorByRange()** method of the **XSpreadsheet** interface. It offers a **gotoStartOfUsedArea()** and a **gotoEndOfUsedArea** method [Gmei08].

A cell range can then be selected via the **XSelectionSupplier** interface offered by the current controller of the model and its **select()** method, which needs the range to be selected as parameter. Additionally, a cell range can be assigned a name, which is saved in the properties of the Calc document [Gmei08].

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

oController = xdoc~XModel~getCurrentController
xSheet = oController~XSpreadsheetView~getActiveSheet()

/*get a range and its address*/
xCellRange = xSheet~getCellRangeByName("B4:C10")
stAddress = xCellRange~XCellRangeAddressable~getRangeAddress()

/*within the address, fill values in the first and the last cell*/
xFirstCellRow = stAddress~bsf.getFieldValue("StartRow")
xFirstCellColumn = stAddress~bsf.getFieldValue("StartColumn")
call uno.setCell xSheet, xFirstCellColumn, xFirstCellRow, "First cell of normal range."

xLastCellRow = stAddress~bsf.getFieldValue("EndRow")
xLastCellColumn = stAddress~bsf.getFieldValue("EndColumn")
call uno.setCell xSheet, xLastCellColumn, xLastCellRow, "Last cell of normal range."

/*we also set an additional cell value*/
call uno.setCell xSheet, 0, 2, "Extra cell."

/**retrieve the complete cell range in use and select it**/
/**from Gmeiner2008**/
xCursor = xSheet~createCursor
xCursor~XUsedAreaCursor~gotoStartOfUsedArea(.false)
xCursor~XUsedAreaCursor~gotoEndOfUsedArea(.true)

oSelection=xdoc~XModel~getCurrentController()~XSelectionSupplier~select(xCursor)

/**assign an area a name**/
/**from Gmeiner2008**/
/*we need a relative reference cell*/
xCell = uno.getCell(xSheet, "A1")
stAddress = xCell~XCellAddressable~getCellAddress

/*as a name has to be unique, we create a timestamp*/
sDate = .DateTime~new~~fullDate~string

oNamedRanges = xdoc~XPropertySet~getPropertyValue("NamedRanges")
xNamedRanges = oNamedRanges~XNamedRanges
xNamedRanges~addNewByName("Named Area" sDate,"A1:C10",stAddress,0)
xdoc~XPropertySet~setPropertyValue("NamedRanges",oNamedRanges)

/*retrieve cell range by name*/
stAddress = xSheet~getCellRangeByName("Named Area" sDate)~XCellRangeAddressable~getRangeAddress()
xFirstCellRow = stAddress~bsf.getFieldValue("StartRow")
xFirstCellColumn = stAddress~bsf.getFieldValue("StartColumn")
xLastCellRow = stAddress~bsf.getFieldValue("EndRow")
xLastCellColumn = stAddress~bsf.getFieldValue("EndColumn")

.bsf.dialog~messageBox("NamedRange start: Column:" xFirstCellColumn "Row:" xFirstCellRow
    .ENDOFLINE -
    "NamedRange end: Column:" xLastCellColumn "Row:" xLastCellRow)

::requires UNO.cls
```

Listing 2.7: litex05_CellRanges. Shows how to access Calc cells [Aham05].

3. Selected Pitonyak OpenOffice.org Writer Examples

The following examples use functionalities specifically available in the Writer application. They include selections, cursor management and text field operations.

3.1. Information on Selections

As Writer has many variants of selections, a script is useful which distinguishes between them.

3.1.1. Example 01 – Selection Types

OpenOffice.org Writer selection retrieval methods, which are `getCurrentSelection()` of the model and `getSelection()` available via the **`XSelectionSupplier`** interface of the document controller, return an "any" object containing either an enumeration of selected items or the selected item itself. As further manipulation highly depends on the object type, the determination of the selection types is very important. They can be one out of six in OpenOffice 3.0 [Sun07, p. 627].

- Text: Potentially a multiple selection, thus a collection supporting the **`XIndexAccess`** interface pointing to the selected text.
- Table Cells: Contains various table cells, returns an `TextTableCursor` service.
- Text Frame: Includes only one text frame of type `com.sun.star.text.TextFrame`.
- Graphic Object: In this case, the selection points to one single object supporting the `com.sun.star.text.TextGraphicObject` service.
- OLE Object: A link to a `com.sun.star.text.TextEmbeddedObject` service is returned.

- Shapes and Forms: The selection is a `com.sun.star.drawing.ShapeCollection` container, a collection of shapes.

Listings 3.1–3.3 examplify how these types can be differentiated and give short information on the selected items. For a working example, all three parts have to be combined to one document. It partially incorporates Pitonyak's text selection macro [Pito04, p. 289–290]. Ideally, a document with some content is needed to execute the macro, and any type of element should be selected.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*retrieve the current selections*/
oController = xdoc~XModel~getCurrentController()
oSelections = oController~XSelectionSupplier~getSelection()

/*object existent? - might happen for Draw, Impress etc.*/
if oSelections ==.nil then
do
   .bsf.dialog~messageBox("Nothing selected.")
   EXIT
end

/*access all selections*/
aSupportedInterfaces = uno.getInterfaceNamesViaReflection(oSelections)~makeArray(" ")
s = "Selection is of type"
```

Listing 3.1: 01_SelectionTypes.rex, first part. It shows how selections can be retrieved.

After a minimal header, listing 3.1 retrieves the current selection via the **`XSelectionSupplier`** interface implemented in all document controllers and the **`getSelection`**`()` method. If a **`.nil`** object is returned, the script exits. Otherwise, the macro accesses the supported interfaces of the selection object via the **`uno.getInterfaceNamesViaReflection`**`()` method.

```
/*possibilities: Text, Table Cells, Text Frame, Graphic Object, OLE Object, Shape*/
/*is text? - supports XIndexAccess*/
if aSupportedInterfaces~hasItem("com.sun.star.container.XIndexAccess") then
do
   s = s "text." .ENDOFLINE
   anythingSelected = .false
   xIndexAccess = oSelections~XIndexAccess
   size = xIndexAccess~getCount()
   if size== 0 then
   do
      .bsf.dialog~messageBox("Nothing selected.")
      EXIT
   end
   /*ok, let's have a closer look on the selections*/
   counter = 1
   do i=1 to size
     oSel = xIndexAccess~getByIndex(i-1)
     oCursor = oSel~XTextRange~getText()~createTextCursorByRange(oSel~XTextRange)
     if oCursor~isCollapsed() == .false then
     do
       anythingSelected = .true
       s = s counter||"." oSel~XTextRange~getString() .ENDOFLINE
       counter = counter+1
     end
   end
```

```
26
    if \ anythingSelected then
28  do
      .bsf.dialog~messageBox("Nothing selected. Only cursor. Exiting.")
30    EXIT
    end
32 end
```

Listing 3.2: 01_SelectionTypes.rex, second part. The case “text selected”.

By OpenOffice 3.0, the interface **XIndexAccess** is implemented by the object returned if and only if the selection consists of at least one text range, thus normal text [Sun07, p. 627]. Therefore, the script asks whether the list of supported interfaces, which is retrieved in listing 3.1, contains the **XIndexAccess** interface (line 3). If so, we assume a sequence of text parts to be selected, to be more precise, objects supporting the **XTextRange** interface.

The **XIndexAccess** interface allows for an iteration over the selected parts, if the selection is empty, the script exits. Otherwise, we retrieve one selection each using **getByIndex()**. Before the actually selected text is extracted and written to the information popup, the macro has to ensure that the current text is not empty. An empty **XTextRange** is consistent with the specifications, in this case the beginning and the ending of a selection is identical, which applies for a cursor [Openb, text/XTextRange]. Similar to Pitonyak, a text cursor service is created over the whole range of the object via the **createTextCursorByRange()** method available in the **XSimpleText** interface (line 18) [Pito04, p. 290]. The **XTextCursor** interface of the cursor offers the **isCollapsed()** method indicating the status of the cursor in dependence of its size. Only if the cursor is not collapsed, the text is retrieved via the **getString()** method of the **XTextRange** interface and added to the message box presented at the end. This complicated retrieval is necessary due to the string size limitation in ooBasic used by Pitonyak, where for large documents the simple check for a zero-sized return of this method might be misleading [Pito04, p. 289].

Typically, even if nothing is selected, at least one collapsed item is present [Pito04]. Resultingly, the selection might implement **XIndexAccess** and contain one item although nothing is highlighted. In this case, `anythingSelected` is **.false** and the script exits (line 27).

```
  else if uno.supportsService(oSelections,"com.sun.star.drawing.ShapeCollection") then
2 do
    iSelCount = oSelections~XShapes~XIndexAccess~getCount()
4   s = s "shape." .ENDOFLINE
    s = s iSelCount "shape(s) selected." .ENDOFLINE
6   aTypes = .table~new()
    do i=1 to iSelCount
8     oShape = oSelections~XShapes~XIndexAccess~getByIndex(i-1)
      sType = oShape~XShapeDescriptor~getShapeType()
10    if aTypes~hasIndex(sType) then
        aTypes~put(aTypes~at(sType)+1,sType)
12    else
        aTypes~put(1,sType)
14  end
    aIndexes = aTypes~allIndexes
```

```
16
    s = s "These are of types" .ENDOFLINE
18  do i=1 to aIndexes~items
      item = aIndexes~at(i)
20    s = s item||":" aTypes~at(item) .ENDOFLINE
    end
22 end
   else if uno.supportsService(oSelections,"com.sun.star.text.TextTableCursor") then
24 do
    s = s "text table cells." .ENDOFLINE
26  s = s "Range selected:" oSelections~XTextTableCursor~getRangeName()
   end
28 else if uno.supportsService(oSelections,"com.sun.star.text.TextEmbeddedObject") then
   do
30  s= s "embedded object."
    s= s||"Class Identifier:" oSelections~XPropertySet~getPropertyValue("CLSID")
32 end
   else if uno.supportsService(oSelections,"com.sun.star.text.TextFrame") then
34 do
    s = s "text frame." .ENDOFLINE
36  s = s||"Text:" .ENDOFLINE
    s = s||oSelections~XTextFrame~getText()~getString()
38 end
   else if uno.supportsService(oSelections,"com.sun.star.text.TextGraphicObject") then
40 do
    s = s "graphic." .ENDOFLINE
42  s = s "Its URL is" oSelections~XPropertySet~getPropertyValue("GraphicURL")||"."
   end
44 else
   do
46  s = s "unknown."
   end
48
   .bsf.dialog~messageBox(s)
50
   ::requires UNO.cls
```

Listing 3.3: 01_SelectionTypes.rex, third part. Handling of non-text selections.

In listing 3.3, non-text selections are handled. The type-check is based on supported services typical for the selected objects. This check is performed with a `uno.supportsService(o,s)` call, which returns `.true` in case of the object `o` supporting service name `s` (e.g. line 1).

The `com.sun.star.drawing.ShapeCollection` service is supported if one or more shapes are selected (line 1). Shapes can be of various types ranging from rectangles over graphics to buttons, they can be accessed within the collection via an **`XIndexAccess`** interface and can be counted (`getCount()`). Apart from the number of selected shapes, the macro also lists the types, which are returned by the `getShapeType()` method of the shape's **`XShapeDescriptor`** interface. The various type definitions are added to a table object and counted.

In line 23, the script determines whether table cells have been selected, which support the `com.sun.star.text.TextTableCursor` service. Its **`XTextTableCursor`** interface provides the `getRangeName()` method, which returns a string representation of rows and columns representing the selected top left and bottom right cells in reference to the whole table [Openb, text/XTextTableCursor], this information is added to the popup.

The support of the com.sun.star.text.TextEmbeddedObject service refers to an embedded object, which always has a "CLSID" property, a so-called class identifier, specifying its type (line 28).

The com.sun.star.text.TextFrame service is only supported if the selection is of type text frame (line 33). The script retrieves the text via the **XTextFrame** interface and the getText() method, which returns an **XText** allowing to call getString(). This finally accesses the text of the frame, other frame content like graphics or tables is ignored.

Another possibility is the selection of a graphic, which is represented by an object supporting the com.sun.star.text.TextGraphicObject service as shown in line 39. It needs a "GraphicURL" property referencing the graphic displayed, which the script retrieves.

If the returned object type is not supported by the script, it finally states that the type is unknown (line 44).

3.2. Cursors and Document Subpart Access

Although text and view cursors have already been utilized from ooRexx before, there are many additional examples showing the combination of both variants and their applications.

3.2.1. Example 02 – Line Breaks In Paragraph

Pitonyak presents an example demonstrating the simultaneous use of both view and text cursor types (see 2.2.1) and their specifities in one OOBasic macro [Pito04, p.286, listing 18], which is transcribed into ooRexx in listing 3.4. When executing the macro, after each line in the paragraph a line break is inserted. At the end, the view cursor is reset to its original position. The user can invoke the helper script A.1 in section A.1 to insert some text into an empty Writer document, which is needed for this script. The result is shown in figure 3.1.

```
/*get the important UNO proxy objects*/
xSc = uno.getScriptContext()
xDt = xSc~getDesktop()
xDoc = xSc~getDocument()

/*get the text*/
oText = xDoc~XTextDocument~getText()
xController = xDoc~XModel~getCurrentController()

/*get the view cursor to determine line ends*/
oViewCursor = xController~XTextViewCursorSupplier~getViewCursor()

/*get the paragraph text cursor to determine paragraph ends*/
oTextCursor = oText~createTextCursorByRange(oViewCursor)~XParagraphCursor
```

```
16 /*we want to restore the view cursor afterwares*/
   oSaveCursor = oText~createTextCursorByRange(oViewCursor)
18
   /*move cursor to start of current paragraph to process the whole*/
20 if \ oTextCursor~isStartOfParagraph() then
   do
22   oTextCursor~gotoStartOfParagraph(.false)
     oViewCursor~gotoRange(oTextCursor,.false)
24 end

26 /*for each line in paragraph*/
   do forever
28   /*the view cursor now knows where the line ends*/
     oViewCursor~XLineCursor~gotoEndOfLine(.false)
30
     /*check whether the end of the paragraph has been reached - then exit*/
32   oTextCursor~gotoRange(oViewCursor,.false)

34   if oTextCursor~isEndOfParagraph() then do
       /*end of paragraph reached - exit*/
36     leave
     end
38   else do
       /*if not - insert a linebreak*/
40     oText~insertControlCharacter(oViewCursor,bsf.getConstant("com.sun.star.text.ControlCharacter"
           ,"LINE_BREAK"),.false)
       oViewCursor~goRight(1,.false)
42
     end
44 end

46 /*restore the view cursor position*/
   oViewCursor~gotoRange(oSaveCursor,.false)
48
   /*load module*/
50 ::requires UNO.CLS
```

Listing 3.4: 02_LineBreaksInParagraph.rex. A line break is added after each line.

In listing 3.4, line 7, the text is retrieved (method `getText()` of the **`XTextDocument`** interface). In this nutshell example as well as in all others of this work, the existence of this interface isn't checked, thus a Writer document execution is assumed. `getText()` returns a `Text` service, which exports **`XText`** [Openb, Text]. Line 8 then ensures access to the **`XController`** of the current document using the `getCurrentController()` method of the document model representing the component [Openb, XModel].

Then the view cursor is accessed via the **`XTextViewCursorSupplier`** interface of the controller which has a `getTextViewCursor()` method. This has already been described [Flat05b, Sun07]. We also create two text cursors (see section 2.2.1 for details) with the first iterating through the current paragraph, which is obtained via creating the cursor in dependence on the view cursor position, and the second saving the current position of the view cursor, which is later required to reset it.

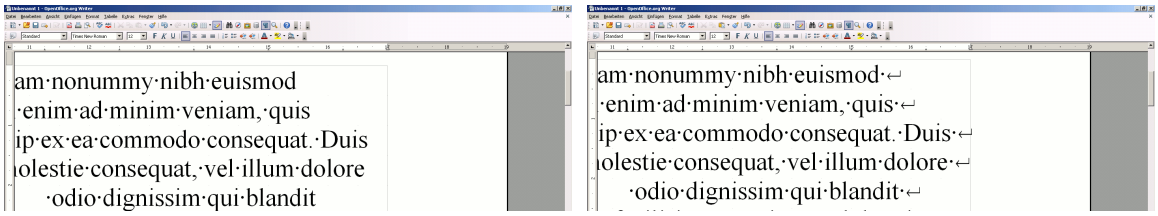
As we intend to traverse through the whole paragraph, at first the cursor position relative to the paragraph has to be determined by the text cursor's `isStartOfParagraph()` method in line 20. Subsequently, both cursors are set to the start by calling the `gotoStartOfParagraph()`

method. As the text view cursor does not implement methods related to the actual text like paragraphs, it is simply set to the text cursor position via `gotoRange(oTextCursor,.false)` in line 23, too.

The following iteration (lines 27–44) puts a line break character at the end of each line, which is found by the view cursor's `gotoEndOfLine()` method, available via the `XLineCursor` interface. The text cursor follows, as it is needed to determine whether the end of paragraph has already been reached.

The line break character is finally inserted with `insertControlCharacter()`, which allows to insert a single character as defined in `com.sun.star.text.ControlCharacter` [Sun07].

In the last step, the visible view cursor is put to its original position saved in the second text cursor.



(a) before (b) after

Figure 3.1.: Text inserted via listing A.1 and selected before (3.1(a)) and after (3.1(b)) executing listing 3.4. The linebreaks at the end of each line are clearly visible.

3.2.2. Example 03 – Remove Duplicate Spaces

Typically, when writing a large document, corrections of text parts occur frequently. During this work, inattention might lead to extra blank spaces, which to remove is a time consuming task. Subsequently, Pitonyak offers a macro solution replacing runs of white-space with a single white-space character [Pito04, p. 293]. Invoking listing A.1 in section A.1 inserts text with some errors into the current Writer document.

According to Pitonyak, "white space" compasses

- tabs (ASCII character 9),
- regular white spaces (ASCII character 32),
- nonbreaking spaces (ASCII character 160),
- start of line (ASCII character 0),

- new paragraphs (ASCII character 13) and
- new lines (ASCII character 10).

As two or more subsequent occurances of these characters have to be reduced to one, the script has to decide which to be retained, thus a ranking is required. Pitonyak suggests the following ranking: start of line > paragraph > line > tab > nonbreaking > normal. For example, if a space is followed by a tab, the space is ranked inferior and deleted.

In our example, we also implement the text selection procedure inspired by Pitonyak [Pito04, p. 294], which is represented as a `TextRange` in OpenOffice.org. We traverse all selected ranges of size bigger than zero, as a zero-sized range contains no character (see section 2.2.1). Additionally, it may refer to a selected image or button within the document, which also can't be used as an input for the script (compare [Pito04, p. 294]). If no useful selection is present, the whole document is traversed.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*check if anything is selected*/
oSelection = xdoc~XModel~getCurrentSelection()

xIndexAccess = oSelection~XIndexAccess
size = xIndexAccess~getCount()

oSelection_noncollapsed = .array~new

/*check whether the selections are collapsed - we use a text cursor that covers the range*/
do i=1 to size
  oSel = xIndexAccess~getByIndex(i-1)~XTextRange
  oCursor = xdoc~XTextDocument~getText()~createTextCursorByRange(oSel)
  if \ oCursor~isCollapsed() then
  do
    oSelection_noncollapsed~append(oSel)
  end
end

/*if we have no uncollapsed selection, we take the whole document as selection*/
if oSelection_noncollapsed~items == 0 then
do
  if .bsf.dialog~dialogBox("No selection found. Traverse whole document?", ,"Warning","YesNo")
      then
    EXIT

  xDocTextRange = xdoc~XTextDocument~getText()~XTextRange

  if \ xdoc~XTextDocument~getText()~createTextCursorByRange(xDocTextRange)~isCollapsed then
    oSelection_noncollapsed~append(xDocTextRange)
  else
  do
    .bsf.dialog~messageBox("You seem to have provided an empty text. Exiting.")
    EXIT
  end
end
```

Listing 3.5: 03_RemoveEmptySpaces.rex, first part. File header and selection determination section.

Listing 3.5 shows the selection determination procedure carried out after the usual header lines. We first obtain access to the selection via the `getCurrentSelection()` method of the **`XModel`** interface of the document, which then provides an **`XIndexAccess`** to retrieve the selected parts. Alternatively, we could use the `getSelection()` method provided by the **`XSelectionSupplier`** interface of the document controller [Sun07, p. 437]. These selections are then iterated over and put into the array `oSelection_noncollapsed` if they are non-empty. This is checked using the **`XTextRange`** interface of the current selection, for which a text cursor is created. The text cursor then supports the method `isCollapsed()`, which is true if the selection is empty (lines 7–22).

If nothing is selected in terms of our definitions, we take the whole document as input. Therefore, the **`XTextRange`** interface of the document's text provides the suitable text range which is then added to the array.

After the traversal array was set up, a text cursor retrieves each character of each selection (listing 3.6).

```
  /*we now check all XTextRanges in the oSelection_noncollapsed array for white spaces*/
2 /*note that for clarity we don't perform a direction check here for the XTextRanges introduced in
      example 03, although it would be useful*/
  size = oSelection_noncollapsed~items
4 do i=1 to size
    oSel = oSelection_noncollapsed~at(i)
6   /*obtain a text cursor over the range*/
    oLeftRange = oSel~getStart()
8   oRightRange = oSel~getEnd()
    oLeftCursor = xdoc~XTextDocument~getText~createTextCursorByRange(oLeftRange)
10  oRightCursor = xdoc~XTextDocument~getText~createTextCursorByRange(oRightRange)
    /*go to start of cursor and "initialize" it*/
12  oLeftCursor~collapseToStart()
    oLeftCursor~goRight(0,.false)
14
    /*we now set the first ASCII character*/
16  oLeftCursor~goRight(1,.true)
    iPrevious = oLeftCursor~XTextRange~getString()
18  iPrevious = convertToASCII(iPrevious)

20
    /*collapse the selection to the end
22   *this is save for the first time, as the selection is >0, thus at least 1*/
    oLeftCursor~collapseToEnd()
24
    /*traverse through the rest of the selection*/
26  do while xdoc~XTextDocument~getText()~XTextRangeCompare~compareRegionEnds(oLeftCursor~
      XTextRange, oRightCursor~XTextRange)
      oLeftCursor~goRight(1,.true)
28    iCurrent = oLeftCursor~XTextRange~getString()
      iCurrent = convertToASCII(iCurrent)
30    /*let's get the rank results*/
      toDel = rankChar(iPrevious iCurrent)
32    /*if we get a result, delete the specific character of lower importance*/
      if toDel<>0 then
34    do
        if toDel==1 then
36      do
          /*delete the current character*/
38        oLeftCursor~XTextRange~setString("")
        end
40      else
```

```
        do
42        /*delete the previous character*/
          oLeftCursor~goLeft(2,.true)
44        oLeftCursor~XTextRange~setString("")
          oLeftCursor~goRight(1,.false)
46        iPrevious = iCurrent
        end
48    end
      else
50    do
        iPrevious = iCurrent
52      oLeftCursor~collapseToEnd()
      end
54  end
end
```

Listing 3.6: 03_RemoveEmptySpaces.rex, second part. The text range traversion section.

To retrieve each character (listing 3.6), at first the start and the end of each selection, an **XTextRange**, is determined using the **getStart()** and **getEnd()** functions of the **XTextRange** interface implemented by the selection (for details on implemented methods, see [Sun07, p. 558]). For these ranges, one cursor is obtained each. To definitely get the starting character, the start cursor is collapsed to its start (**collapseToStart()**) and "initialized" by moving it 0 characters to the right without selecting (line 13). Pitonyak proposes this step every time before actually moving the cursor to set its correct moving direction [Pito07, Pito04, p. 169 and p. 291, respectively].

In contrast to Pitonyak who automatically sets the first character to ASCII code 0, we also retrieve it within an extra step before the iteration, which at any case is more precise. Subsequently, the cursor is shifted one character to the right selecting it, the string is retrieved with **getString()** and converted to the corresponding ASCII-code with the custom routine `convertToASCII()`. Before progressing with traversing, the cursor is collapsed to its end, thus altogether moved one position further to the right within the selection. This step is repeated until the end of the selection under investigation is reached. The **compareRegionEnds(r1,r2)** method of the **XTextRangeCompare** interface compares the positions of two text ranges `r1,r2`. It returns 1 if `r1` ends before `r2`, -1 otherwise and a 0 if they have the same position [Openb, **XTextRangeCompare**].

The next character is thus obtained as long as the condition is true, and both the previous and the current character are parameters of the `rankChar()` routine call (see listing 3.7). The routine ranks the supplied characters and returns either -1 if the previous character has to be retained or 1 otherwise. In case of not both characters being white space, 0 is given back and no position is eliminated. `rankChar()` uses the `isWhiteSpace()` routine (lines 13–24) determining the weight of the current character on the basis of an array of ASCII codes, which indices reflect the character's rank.

The script proceeds in line 33 of listing 3.6 in dependence on the results of `rankChar()` saved in the variable `toDel`. If `toDel` equals 1, we can delete the character still selected by setting it to an empty string. If the previous character is less important, the cursor has to move back 2 positions with the selecting option set to `.true`, which causes a selection inversion on the current character and a selection of the one before, which is deleted subsequently. The script continues and the cursor is moved to the right, but just one character as the second has been removed.

```
::REQUIRES UNO.cls

::ROUTINE convertToASCII
  PARSE ARG ascii
  /*it is possible that the retrieved string is length zero - e.g. if an image is selected - or
      of length x>1 if a field is selected as did Pitonyak, we then set the value to 65*/
  if ascii~length <> 1 then
    ascii = 65
  else
    ascii = c2d(ascii)

RETURN ascii

/*routine isWhiteSpace returns a 0 if the character is not defined as a white space character and
     the position within the chain if it is. 1 defines the lowest, size(x) the highest priority*/

::ROUTINE isWhiteSpace
  PARSE ARG char
  whites = .array~of(0,13,10,9,160,32)
  ret = 0
  if whites~hasItem(char) then
  do
    ret = whites~index(char)
  end


RETURN ret

/*rankChar takes into account two characters; it returns -1 if the previous has to be deleted, 0
    if this pair has to be ignored and 1 if the current character has to be deleted.*/
::ROUTINE rankChar
  PARSE ARG previous current

  ret = 0

  current_idx = isWhiteSpace(current)
  previous_idx = isWhiteSpace(previous)
  if current_idx>0 & previous_idx>0 then
  do

    if current_idx < previous_idx then
    do
      ret = -1
    end
    else
    do
      ret = 1
    end
  end

RETURN ret
```

Listing 3.7: 03_RemoveEmptySpaces.rex, part three. The help routines.

It is important to notice that the script does not include text fields, which is a result of their

sources being outside of the document's text scope. Each text field is treated as one character independent of its actual content.

3.2.3. Example 04 – Replace Formatting

Apart from the traditional search-and-replace method dealing with texts, which was already explained in [Aham05] for Writer documents or in [Burg06] for Calc cells, OpenOffice.org offers additional functionalities in its GUI version like a search-and-replace method based on attributes. This variant can also be accessed via a macro by calling the object methods `setSearchAttributes()` and `setReplaceAttributes()` of the search or replace descriptors [Pito04, p. 300]. Their ooRexx usage is explained in [Aham05]. The Pitonyak script transcribed searches for bold text, converts it to a text formatted with normal weight but highlights it with two curly brackets inserted at the beginning and the end. The document thus should contain some text, and some formatted as "bold" as shown in figure 3.2.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

xReplaceDescriptor = xdoc~XModel~XReplaceable

/*create replace descriptor and set its values*/
oReplace = xReplaceDescriptor~createReplaceDescriptor()
oReplace~setReplaceString("{{&}}")
oReplace~setSearchString(".*")

/*set option for the replacement descriptor*/
oReplace~XPropertySet~setPropertyValue("SearchRegularExpression", box("boolean",.true)) --use
    regex
oReplace~XPropertySet~setPropertyValue("SearchStyles", box("boolean",.true)) --we search within
    styles
oReplace~XPropertySet~setPropertyValue("SearchAll",box("boolean",.true)) --for the entire
    document

/*now we define the attribute to find*/
SrchAttributes = uno.CreateArray(.UNO~PROPERTYVALUE,1)
SrchAttributes[1] = uno.createProperty("CharWeight",box("float",bsf.getConstant("
    com.sun.star.awt.FontWeight","BOLD")))

/*create the attribute to replace with*/
ReplAttributes = uno.CreateArray(.UNO~PROPERTYVALUE,1)
ReplAttributes[1] = uno.createProperty("CharWeight",box("float",bsf.getConstant("
    com.sun.star.awt.FontWeight","NORMAL")))

/*assign attributes to the replacement descriptor using the XPropertyReplace-interface*/
oReplace~XPropertyReplace~setSearchAttributes(SrchAttributes)
oReplace~XPropertyReplace~setReplaceAttributes(ReplAttributes)

.bsf.dialog~messageBox(xReplaceDescriptor~replaceAll(oReplace))

::REQUIRES UNO.cls
```

Listing 3.8: 04_ReplaceFormatting.rex. The script searches for certain attributes in the text and replaces them.

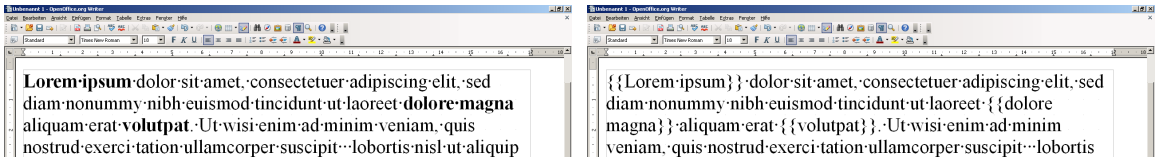
The writer's `XModel` supports the `XReplaceable` interface, an extension of `XSearchable`. Resultingly, the following steps can also be adapted to perform a search-only operation. To perform a search-and-replace method, the replace descriptor `XReplaceDescriptor` has to be created at this point with the `createReplaceDescriptor()` method (line 9. For searching only, the `XSearchDescriptor` would be sufficient. This descriptor can be obtained by calling the `createSearchDescriptor()` method [Sun07, p. 573].

Then the search and the replace strings have to be configured via `setSearchString()` and `setReplaceString()` of the descriptor [Aham05, Sun07]. In our case, we don't look for a certain text, but for attributes. Resultingly, the search string has to match all texts – which is expressed by `".*"` in regular expression patterns. The `.` stands for any character, which is repeated zero to n times ($*$). The replace string `"{{&}}"` then includes the surrounding brackets and an ampersand, which inserts the formerly matched text.

Before defining our search attribute parameters, in lines 13–16 we first have to set various replace descriptor options via the `XPropertySet`. By default, it only text is searched – we have to activate the regular expression pattern mechanism `"SearchRegularExpression"`, as we later need it to embrace the whole match text with curly brackets. Additionally, it is vital to switch on the search styles option `"SearchStyles"`, as this flag is commonly unset. In the nutshell example, all occurrances of bold text will be replaced, therefore we also set the `"SearchAll"` option.

Subsequently, the search attributes array `SrchAttributes` is created, in our case it only compasses the `"CharWeight"` property. The replace attributes are set in the same style.

Both the search and replace descriptor services implement an `XPropertyReplace` interface, which is needed for the description of the search [Sun07, p. 574]. The attributes to search for, `SrchAttributes`, are set via the `setSearchAttributes()` method, the replace attributes via `setReplaceAttributes()`.



(a) before (b) after

Figure 3.2.: Text inserted via listing A.1, some portions were defined as bold. After invoking listing 3.8, the bold text is modified (3.2(b)).

Finally, the replace process can be triggered and controlled using methods inherited from the `XRepleacable` interface, e.g. `findFirst()` [Sun07] or with `replaceAll()`, which processes

the whole text and returns the number of substitutions [Openb, XReplaceable], which is then displayed in a message box.

3.2.4. Example 05 – Replace Selected

In many cases, it is desirable to replace or search text only within a special part of the document. If anything is selected, the GUI therefore allows to narrow the search to the these parts only. Unfortunately, although the **XPropertySet** of the search descriptor allows to specify many options like backwards search, search for complete words only or a similarity search, a flag for limiting the search to selected parts of the document is missing (compare [Openb, SearchDescriptor]). Pitonyak therefore suggests taking a cursor to traverse through the selected parts and combining it with a repeated check whether the end of the selection has been reached [Pito07, p. 212]. While Pitonyak just performs a search, we extend the example by additionally replacing the text with a string entered before. The document the script is invoked upon should be a Writer document containing some text, e.g. after executing listing A.1 of appendix A.1.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*check if anything is selected*/
oSelection = xdoc~XModel~getCurrentSelection()

xIndexAccess = oSelection~XIndexAccess
size = xIndexAccess~getCount()

oSelection_noncollapsed = .array~new

/*check whether the selections are collapsed - we use a text cursor that covers the range*/
do i=1 to size
  oSel = xIndexAccess~getByIndex(i-1)~XTextRange
  oCursor = xdoc~XTextDocument~getText()~createTextCursorByRange(oSel)
  if \ oCursor~isCollapsed() then
  do
    oSelection_noncollapsed~append(oSel)
  end
end

/*if we have no uncollapsed selection, we take the whole document as selection*/
if oSelection_noncollapsed~items == 0 then
do
  if .bsf.dialog~dialogBox("No selection found. Traverse whole document?", ,"Warning","YesNo")
      then
    EXIT

  xDocTextRange = xdoc~XTextDocument~getText()~XTextRange

  if \ xdoc~XTextDocument~getText()~createTextCursorByRange(xDocTextRange)~isCollapsed then
    oSelection_noncollapsed~append(xDocTextRange)
  else
  do
    .bsf.dialog~messageBox("You seem to have provided an empty text. Exiting.")
    EXIT
  end
end
```

```
40
   /*initialization of the search&replace-options*/
42 xSearchDescriptor = xdoc~XModel~XSearchable

44 /*create search descriptor and set its values*/
   oSearch = xSearchDescriptor~createSearchDescriptor()
46 oSearch~setSearchString(.bsf.dialog~inputBox("Enter text to be searched.",,"question"))

48 sReplace = .bsf.dialog~inputBox("Enter replacement text.",,"question")

50 /*set option for the search descriptor*/
   oSearch~XPropertySet~setPropertyValue("SearchCaseSensitive",box("boolean",.false))
52
   /*we now check all XTextRanges in the oSelection_noncollapsed array for white spaces*/
54 /*note that for clarity we don't perform a direction check here for the XTextRanges introduced in
        example 03, although it would be useful*/
   size = oSelection_noncollapsed~items
56 do i=1 to size
     oSel = oSelection_noncollapsed~at(i)
58
     /*obtain a text cursor over the range*/
60   oLeftRange = oSel~getStart()
     oRightRange = oSel~getEnd()
62   oLeftCursor = xdoc~XTextDocument~getText~createTextCursorByRange(oLeftRange)
     oRightCursor = xdoc~XTextDocument~getText~createTextCursorByRange(oRightRange)
64   /*go to start of cursor and "initialize" it*/
     oLeftCursor~collapseToStart()
66   oLeftCursor~goRight(0,.false)

68   /*we now search for the next occurance after the text cursor*/
     vFound = xSearchDescriptor~findNext(oLeftCursor,oSearch)
70
     do while vFound<>.nil
72     if xdoc~XTextDocument~getText()~XTextRangeCompare~compareRegionEnds(vFound~XTextRange,
           oRightCursor~XTextRange)<>-1 then
       do
74       vFound~XTextRange~setString(sReplace)
         vFound = xSearchDescriptor~findNext(oLeftCursor,oSearch)
76     end
       else
78     leave
     end
80
   end
82
   ::REQUIRES UNO.cls
```

Listing 3.9: 05_ReplaceSelected.rex. A search-and-replace procedure is called for a selection only.

The first part of the script shown in listing 3.9 (lines 1–39) is identical to listing 3.5, it checks for non-empty selections and takes the whole document if nothing has been chosen. The selected text ranges are contained in `oSelection_noncollapsed`.

Then a search descriptor is created, which is similar to a replace descriptor and was introduced for ooRexx by Burger [Burg06]. The **`XSearchable`** interface implemented by the writer document model offers a **`createSearchDescriptor`**`()` method, which returns the descriptor service `oDescriptor` (line 45). The **`setSearchString`**`()` method is again used to set the string to search, here it is taken from an **`inputBox`** dialog. The replace string is just stored in the variable `sReplace`. The descriptor is specified as non-case sensitive, of course other properties could be applied as well.

In this case, we have to define a search descriptor and replace the text manually. This is caused by a replace descriptor being only useful if all occurrances within the whole text have to be substituted using its `replaceAll()` method, which is not the case when concentrating on selected parts only [Pito07, p. 211].

The script then iterates over all selected subparts and creates two cursors, one covering the left edge of the selection (`oLeftCursor`), one the right edge (`oRightCursor`). The left-edge cursor is collapsed to its start and "initialized" as proposed by Pitonyak [Pito07, p. 169]. The script proceeds similar to listing 3.8 with a `compareRegionEnds(R1,R2)` method accepting two text ranges as parameters. It checks whether the next position found using the descriptor's `findNext()` method is still within the selected range. If it is, the search string is replaced. This step is repeated until either the end of the selection is reached or the string can't be detected anymore within the text, in this case `findNext()` returns `.nil`.

A selected text is replaced, but is the new string to be included in the selection? Writer behaves ambiguously: In principle, Writer adds it to the current selection. But is the matching position at the beginning or the end of the original selection, the new string is not included and the original selection ranged changes, which has to be taken into account.

3.3. Text Fields

Some introductory examples on how to manage text fields as well as some more advanced scripts of practical relevance are presented. Text field masters and their usage and connections to text fields are introduced, and one example shows how a form letter containing various fields can be created and executed within OpenOffice.org Writer.

3.3.1. Example 06 – Create Fields

The most basic assignment is inserting a text field and modifying its properties, an example is shown in section 2.2.3. Pitonyak also presents such a simple script in [Pito04, p. 320], but additionally he integrates a specially newly created date and time format and he manually modifies the presented date. Listing 3.10 demonstrates the resulting ooRexx script. We reset the year by -10 and create the new format `"YYYY-MMMM-DD HH:MM:SS"`, if it is not existent. The script can be invoked on any Writer document.

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
```

```
xMSF = xdoc~XMultiServiceFactory

oText = xdoc~XTextDocument~getText()

/****datetime text field****/
/*we want to have a special number format - create a new one*/
oNumberFormats = xdoc~XNumberFormatsSupplier~getNumberFormats()

/*if the format is existent - fine, else create it*/
sNewFormat = "YYYY-MMMM-DD HH:MM:SS"
/*we have to define the locale*/
oLocale = .bsf~new("com.sun.star.lang.Locale")
oLocale~bsf.setFieldValue("Country","USA")
oLocale~bsf.setFieldValue("Language","en")
iFormatIndex = oNumberFormats~XNumberFormats~queryKey(sNewFormat,oLocale,.true)

if iFormatIndex = -1 then
  iFormatIndex = oNumberFormats~addNew(sNewFormat,oLocale)

/*explanatory text*/
oText~XSimpleText~insertString(oText~XTextRange~getEnd(),"Today is ",.false)
/*create the datetime text field*/
oField = xMSF~createInstance("com.sun.star.text.textfield.DateTime")
/*set properties*/
oField~XPropertySet~setPropertyValue("IsFixed",box("boolean",.true))
oField~XPropertySet~setPropertyValue("NumberFormat",box("integer",iFormatIndex))

/*insert into text*/
oText~insertTextContent(oText~XTextRange~getEnd(),oField~XTextContent,.false)

/*modify the date*/
oDateTime = oField~XPropertySet~getPropertyValue("DateTimeValue")
iYear = oDateTime~bsf.getFieldValue("Year")-10
oDateTime~bsf.setFieldValue("Year",iYear)

oField~XPropertySet~setPropertyValue("DateTimeValue",oDateTime)
xdoc~XTextFieldsSupplier~getTextFields()~XRefreshable~refresh()

::REQUIRES UNO.cls
```

Listing 3.10: 06_CreateFields.rex. A field is created and refreshed after a modification.

After the usual header lines, script 3.10 first starts with ensuring that the necessary date-time format is already present (lines 10–22). Therefore, all available formats are retrieved as `com.sun.star.util.NumberFormats` using the `getNumberFormats()` method supplied by the **`XNumberFormatsSupplier`** interface of the model [Sun07, p. 518]. The number formats service specifies a container of number formats and allows for access and modifications via its interfaces **`XNumberFormats`** and **`XNumberFormatTypes`** [Openb, NumberFormats].

The new format is created, it later acts as a query object. It consists of a format string and a locale, which is of type `com.sun.star.lang.Locale` and defines the format's country and language via its fields. Similar to Pitonyak, in line 19 a search within the returned formats using the `queryKey()` method provided by the **`XNumberFormats`** interface of the number formats container is performed. It returns either the index number of the format within the container or -1 if the string can't be detected. In this case, the `addNew()` method expecting the format as well as the locale as parameters is called to create the new format; the method returns the format's new index.

After inserting some explanatory text and the field after its creation (see section 2.2.3), the `"isFixed"` property is set to `.true`. This causes the date-time text field not to always show the current date [Openb, DateTime], which is necessary as we intend to modify it. In the same way, the number format is changed, which is done via its index number in the number formats container.

The date and time of the field is saved in the `"DateTimeValue"` property, which contains a `com.sun.star.util.DateTime` object. To modify it, we first retrieve it (`getPropertyValue()`, line 36), extract the `"Year"` field of the object, change it and set the field to the new value. Similarly, the field property is set. A modification of the date-time property is only possible after inserting the field to the document. Resultingly, the field primarily shows the current date-time values before they get modified, and it does not change its textual representation even after a field update. To avoid a discrepancy, a refresh is necessary, at least for the use of fields and their properites in ooRexx. Pitonyak does not perform this refresh in his sample script [Pito04, p. 321]. Interestingly, the field itself does not provide any possibility of an alignment with the property sources, the offered `update()` method of the fields's `XUpdatable` interface only sets the date-time value to the current system date [Sun07, p. 592]. Nonetheless, the `com.sun.star.TextFields` container (see listing 3.11) implements the `XRefreshable` interface and the `refresh()` method, which updates the text field as expected (line 41).

3.3.2. Example 07 – Display Fields

In some cases, it might be important to access all text fields of a document and retrieve their textual representations. Pitonyak gives an example on how to extract basic information from the field, e.g. its type. Furthermore, he includes a necessary special treatment for the `com.sun.star.TextField.Annotation` field [Pito04, p. 317], which ooRexx usage has already been demonstrated by Prem [Prem06]. The script's final output is a message box, where the type and the content of each text field is listed (see listing 3.11). To get a useful result, this script should be executed from within a Writer document containing text fields, e.g. after invoking listing 3.10.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*get the fields out of the text*/
oTextFields = xdoc~XTextFieldsSupplier~getTextFields()
oEnum = oTextFields~XEnumerationAccess~createEnumeration()

/*iterate over fields and extract information*/
/*store in variable s*/
s=""
do while oEnum~hasMoreElements()
```

```
14    oField = oEnum~nextElement()
      s = s oField~XTextField~getPresentation(.true) "=" --returns command of the field
16    if oField~XServiceInfo~supportsService("com.sun.star.text.TextField.Annotation") then
      do
18      /*note that an annotation does not have a displayed content,
         *getPresentation(.false) therefore returns an empty string.
20       *we therefore obtain the author and the content*/
        sAuthor = oField~XPropertySet~getPropertyValue("Author")
22      sContent = oField~XPropertySet~getPropertyValue("Content")
        s = s sAuthor "says" sContent
24    end
      else
26    do
        s = s oField~XTextField~getPresentation(.false)
28    end

30    s = s .ENDOFLINE
    end
32
    .bsf.dialog~messageBox(s)
34
    ::REQUIRES UNO.cls
```

Listing 3.11: 07_DisplayFields.rex. The script shows all text fields present in the document.

A container with all text fields can easily be obtained via the `XTextFieldsSupplier` interface provided by the model and its `getTextFields()` method. This container does not only allow to refresh the fields (see listing 3.10), but also implements the `XEnumerationAccess` interface, which gives access to the single fields. `createEnumeration()` returns an `XEnumeration`, which can be iterated over using the `hasMoreElements()` methods (see lines 13–31).

In each step, the current field is retrieved from the container, and via its `XTextField` interface, the `getPresentation()` method is called. This method expects one boolean parameter, if this is `.true`, the type of the field is returned, otherwise the method returns the textual representation [Pito04, p. 317].

Unfortunately, for the annotation text field, the method returns an empty string, as it does not have a textual representation, text and author information is available via the `XPropertySet`, which includes an `"Author"`, a `"Content"` as well as a `"Date"`/`"DateTimeValue"` property holding the respective values.

We therefore check for the annotation service to be supported by the current field, which can be done using the `supportsService()` method of the `XServiceInfo` interface (line 16), which all text fields implement. In case of an annotation, we simply read out the property values, otherwise we obtain the textual representation via the `getPresentation()` method.

3.3.3. Example 08 – Create Field Masters

Apart from standard text fields, OpenOffice.org additionally offers text field masters. They are connected with normal text fields, which obtain their data from the attached field master

[Pito04, p. 318]. Text field masters are grouped in `com.sun.star.text.FieldMaster.*`. Besides a `User` variant displaying a variable a `SetExpression` which is sub type of expression, a `DDE` field holding the DDE (Dynamic Data Exchange; a protocol for exchange between programs) command for the DDE text field, a `Database` field responsible for data source definitions and a `Bibliography` field containing bibliography settings are available [Sun07, p. 593].

While normal text fields can only be accessed by enumeration (see listing 3.11), a text field master may also be addressed using its name. Listing 3.12 demonstrates a simple text field master insertion into the text, a similar script is shown by Pitonyak [Pito04, p. 322]. This script works on all Writer documents.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xMSF = xdoc~XMultiServiceFactory

oText = xdoc~XTextDocument~getText()

oMasters = xdoc~XTextFieldsSupplier~getTextFieldMasters()

sName = "ChristophField"
sLead = "com.sun.star.text.FieldMaster.User"

/*if master already exists, special handling
 *otherwise error when setting name - can only be set BEFORE inserting!*/
if oMasters~hasByName(sLead||"."||sName) then
do
  if .bsf.dialog~dialogBox("Master already existent. Delete?", ,"Warning","YesNo") then
  do
    .bsf.dialog~messageBox("Nothing to do. Exiting.")
    EXIT
  end
  else
  do
    oMasterField = oMasters~getByName(sLead||.||sName)
    oMasterField~XComponent~dispose()
  end
end

/*create a user text field that requires a field master*/
oUField = xMSF~createInstance("com.sun.star.text.TextField.User")

/*get for the field master*/
oMasterField = xMSF~createInstance(sLead)

/*set name, it can't bet set or changed after insertion into document*/
oMasterField~XPropertySet~setPropertyValue("Name",sName)

/*set content, the field master tells the field what to display*/
oMasterField~XPropertySet~setPropertyValue("Content","Hello, this is a test field")

/*attach the field master to the field, it is now dependent*/
oUField~XDependentTextField~attachTextFieldMaster(oMasterField~XPropertySet)

/*insert field field*/
oText~insertTextContent(oText~XTextRange~getEnd(),oUField~XTextContent,.false)

::requires UNO.cls
```

Listing 3.12: 08_CreateMasterFields.rex. A text field master and a dependent text field are created.

A text field master creation is more complicated than a simple text field insertion. As a special requirement, the name of the text field master has to be set before it is added to the document, and it can't be changed afterwards. The name should be unique, therefore listing 3.12 checks for already available text field masters, which are returned by the `getTextFieldMasters()` method of the `XTextFieldsSupplier` interface (line 9). The representation is a combination of the type (in this case `com.sun.star.text.FieldMaster.User`, which contains a global document variable which is created and displayed by the `com.sun.star.text.TextField.User` field) and its name. First the proposed unique name of the new master is defined within the variable `sName` and its type in `sLead`, then the text field master container is queried using its `XNameAccess` interface and the `hasByName()` method. If the field is already present, a dialog box offers its deletion, if confirmed, the script proceeds with obtaining the master from the container (`getByName()`) and its removal from the document by executing the `dispose()` method available via the `XComponent` interface of the text field master.

Subsequently, both the text field master and its dependent text field are created by calling the `XMultiServiceFactory`'s `createInstance()` method and the master's name is set. In this user-defined text field master, we also add some content, which the dependent field later displays.

In line 43, the dependent field gets assigned to its master. Each dependent field implements the `XDependentTextField` service, which handles the master–sibling pairs [Sun07, p. 592]. The service has two methods, `attachTextFieldMaster()` which attaches a text field master to the field and must be called, and `getTextFieldMaster()` which returns the currently assigned field master. Each dependent field has only one master [Openb, XDependentTextField].

In the last step, the dependent text field is inserted as usual (compare 2.2.3), in this case at the end of the document.

3.3.4. Example 09 – Display Field Masters

Similar to normal text fields, Pitonyak also implements an example displaying all available field masters of the document [Pito04, p. 319]. The overview output by the script additionally indicates the number of dependent text fields per master.

By default, a document already contains four text field masters, all of type `SetExpression`. They represent the number range field masters for drawings, tables, illustrations and texts as well as their settings, and are responsible for the numbers of the respective objects' captions.

The last part of the script demonstrates the retrieval of name and content of a predefined text field master, `com.sun.star.text.FieldMaster.User.ChristophField`, which is of type user and must be present within the document, e.g. by executing listing 3.12 on the same Writer file before.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()

/*get the field masters out of the text
 *provides an XNameAccess interface as return value!*/
oMasters = xdoc~XTextFieldsSupplier~getTextFieldMasters()
oMasterNames = oMasters~getElementNames()

/*iterate over all field masters
 *DependentField Property is an array of text fields
 *which are dependent on the field master*/
size = oMasterNames~items
s = ""
do i=1 to size
  sMasterName = oMasterNames[i]
  oMaster = oMasters~getByName(sMasterName)
  s = s "***" sMasterName "***" .ENDOFLINE
  s = s oMaster~XPropertySet~getPropertyValue("Name") "contains" (oMaster~XPropertySet~
      getPropertyValue("DependentTextFields"))~items  "dependent fields" .ENDOFLINE
  s = s .ENDOFLINE
end

/*vice versa - directly obtain a field master based on the name*/
/*the user field must be added to the text before*/
field = "com.sun.star.text.FieldMaster.User.ChristophField"
if oMasters~hasByName(field) then
do
  oMaster = oMasters~getByName(field)
  s = s "Directly obtained the field master" oMaster~XPropertySet~getPropertyValue("Name")
      .ENDOFLINE
  s = s "Field contains the text" oMaster~XPropertySet~getPropertyValue("Content")
end

.bsf.dialog~messageBox(s)

::requires UNO.cls
```

Listing 3.13: 09_DisplayMasterFields.rex. Text field masters and the number of its dependent fields.

As for normal text fields, **`XTextFieldsSupplier`** provides access to the text field masters (`getTextFieldMasters()`). The master container implements the **`XNameAccess`** interface, therefore it is able to return the element names of the text field masters as an array. Lines 16–22 iterate over the array items, for each item the text field master is retrieved via the `getByName()` method. Additionally, the number of dependent text fields are counted. A text field master's property, `"DependentTextFields"`, codes the sequence of dependent fields as **`XDependentTextField`** interfaces, which then would allow to access the fields themselves. For our purposes, we only count the number of elements in the sequence and add it to the final output (line 20).

Similarly, it is possible to obtain a text field master's content and name. The container holds all fields, the retrieval can be processed using the `getByName()` method. Name and content

are then stored as for normal text fields within the property set and the properties `"Name"` and `"Content"`.

3.3.5. Example 10 – Fields: Table Numbering

This example demonstrates how the predefined number ranges can be accessed and modified in terms of numbering type or numbering steps. It also shows that it is possible to insert a number field connected to the table numbering range outside of a table caption.

In principle, a text field master is responsible for the number range, the properties of one number is then set via its text field. It can be shown that within one range different numbering types can be existent, nonetheless as soon as OpenOffice.org automatically inserts a caption, all numbering types and formats are adjusted to the style of the GUI dialog.

The script retrieves the text field master of the table caption numbering and then inserts a new text field attached to it into the text. If no table is present, the text field starts with number 1, otherwise it is automatically adapted to the current number range.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xMSF = xdoc~XMultiServiceFactory

/*reference the field master - it is supposed to be existend*/
sMasterFieldName = "com.sun.star.text.FieldMaster.SetExpression.Table"
oMasterField = xdoc~XTextFieldsSupplier~getTextFieldMasters()~getByName(sMasterFieldName)

/*now create the field to insert*/
oField = xMSF~createInstance("com.sun.star.text.TextField.SetExpression")

oField~XPropertySet~setPropertyValue("NumberingType",box("SH",bsf.getStaticValue("
     com.sun.star.style.NumberingType","CIRCLE_NUMBER")))
oField~XPropertySet~setPropertyValue("Content","Tabelle+1")
oField~XPropertySet~setPropertyValue("IsShowFormula",box("boolean",.false))
oField~XPropertySet~setPropertyValue("IsFixedLanguage",box("boolean",.false))

oField~XDependentTextField~attachTextFieldMaster(oMasterField~XPropertySet)

oText = xdoc~XTextDocument~getText()
oText~insertTextContent(oText~XTextRange~getEnd(),oField~XTextContent,.false)
xdoc~XTextFieldsSupplier~getTextFields()~XRefreshable~refresh()

::requires UNO.cls
```

Listing 3.14: 10_FieldExampleTableNumbering.rex. Uusage of predefined number ranges.

At first, the field masters always present in the document for maintaining the table number range, `com.sun.star.text.TextMaster.SetExpression.Table` are retrieved via their names. Then we create the new text field, which has to be of a master-compatible type, `com.sun.star.text.TextField.SetExpression`. Its properties are set. `"NumberingType"` codes the numbering style as a short defined in `com.sun.star.style.NumberingType`. In our

case, we define `"CIRCLE_NUMBER"`, a small number within a circle. This numbering style is only available up to 20. As content of the field we define an expression adding one to the current variable coding the table number, `"Tabelle+1"`. It is very important to adapt this variable according to the language version of OpenOffice.org! In the English version, the variable is `"Table"`, while in the German version it reads `"Tabelle"`.

Then the text field is attached to its field master, inserted and the text fields are refreshed as already described in listing 3.10.

3.3.6. Example 11 – Fields: Form Letter

Form letters are one application heavily depending on text fields. Typically, the text field's sources are databases or spreadsheet files. Apart from fields containing address or name, in many cases form letters also utilize conditional text fields, where the textual representation of a field is dependent on an external condition.

Prem already designed a mail merge script for ooRexx and OpenOffice.org, which is completely based on an iteration in ooRexx inserting data read from a spreadsheet manually [Prem06]. Although this approach is very flexible, as multiple different programs can be combined with the help of ooRexx, it neither uses internal database connections nor the mail merging capabilities of OpenOffice.org.

Our example is completely based on OpenOffice.org's internal functionality, thus closely resembles the manual form letter creation process. In order to execute the script, an auxiliary database "addresses" has to be registered before as a datasource within OpenOffice.org's database context. The table has to contain one "First Name", "Last Name", "Street", "ZIP" and "City" field, all of them defined as strings. The tiny integer field "Sex" codes the people's gender, a 1 stands for a female. In principle, datasource management is also scriptable [Schm07]. Therefore, a generating script is also provided, which cares about creating the database, filling it with some examples and registering it as data source in OpenOffice.org. It can be found in appendix A.1, listing A.2.

The script connects to the datasource, extracts the address data and builds up a form letter including an address section and a conditional field modifying the form of address in dependence of gender. This form letter is then merged with the data and saved. We structure the script in three blocks: The first block creates the address block (listing 3.15), the second inserts an examplary text containing a conditional field (listing 3.16, see figure 3.3) and in listing 3.17 the merger is done and the document saved. Graphical file input: Appendix A.2, listing A.7.

```
/*optional - if the document has not been saved yet*/
saveAs = "c:/temp/formLetter.odt"

/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
oText = xdoc~XTextDocument~getText()
xMSF = xdoc~XMultiServiceFactory

/*the database "addresses" with table "address" must be registered within OOo.org
 *fields: First Name, Last Name, Street, ZIP, City, Sex*/

/*create the masterfields and the textfields*/
dbfields = .array~of("First Name", "Last Name", "Street", "ZIP", "City")
masterfields = .array~new(dbfields~items)
fields = .array~new(dbfields~items)

do i=1 to dbfields~items
  /*field masters*/
  masterfields[i] = xMSF~createInstance("com.sun.star.text.FieldMaster.Database")
  /*only one out of DataBaseName, DataBaseURL, DataBaseResource can be set*/
  masterfields[i]~XPropertySet~setPropertyValue("DataBaseName","addresses")
  masterfields[i]~XPropertySet~setPropertyValue("DataTableName","address")
  masterfields[i]~XPropertySet~setPropertyValue("DataColumnName",dbfields[i])

  /*text fields*/
  fields[i] = xMSF~createInstance("com.sun.star.text.textfield.Database")
  fields[i]~XDependentTextField~attachTextFieldMaster(masterfields[i]~XPropertySet)
  fields[i]~XPropertySet~setPropertyValue("Content","<"||dbfields[i]||">")
end

/*set up the form letter address box*/
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("First Name",dbfields)]~
    XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("Last Name",dbfields)]~
    XTextContent,.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","LINE_BREAK"),.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("Street",dbfields)]~
    XTextContent,.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","LINE_BREAK"),.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("ZIP",dbfields)]~XTextContent,
    .false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("City",dbfields)]~XTextContent,
    .false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)
```

Listing 3.15: 11_FieldExampleFormLetter.rex, first part. Building of the address block.

Listing 3.15 creates an array holding the field masters responsible for the database connection. For each column, a field master of type `com.sun.star.text.FieldMaster.Database` is set in the array. Important properties of this field are `"DataBaseName"` equal to the name of the database within data sources, and `"DataTableName"` allowing to specify the data table as well as `"DataColumnName"`, which refers to the column (lines 22–27). Afterwards, the text fields are created similarly and attached to the respective field master (see listing 3.12). The

`"Content"` property includes the value presented in the form letter. According to [Openb, test/textfield/Database], before the first merger this property should be equal to the column embraced with parenthesis, but examples prove it to be initially empty when inserted via script, which is contrary to normal GUI insertions. To mimicry the default behaviour, we manually set the `"Content"` property (line 32).

In lines 35–46, the text fields are put into the document with the help of cursor operations and the `insertTextContent()` method. Although there is no `"Content"` property for the conditional field, the `"CurrentPresentation"` enables the initial setting of the textual representation [Openb]. The routine `getField` (e.g. line 36) simply returns the index of an argument `name` within the array `dbfields`. Its source code is available in listing 3.17.

```
/*insert a conditional field*/
oCondField = xMSF~createInstance("com.sun.star.text.textfield.ConditionalText")
oCondField~XPropertySet~setPropertyValue("TrueContent","Miss")
oCondField~XPropertySet~setPropertyValue("FalseContent","Mister")
oCondField~XPropertySet~setPropertyValue("Condition","addresses.address.Sex==1")
oCondField~XPropertySet~setPropertyValue("CurrentPresentation","<SexConditional>")

/*last name field*/
oLNameField = xMSF~createInstance("com.sun.star.text.textfield.Database")
oLNameField~XDependentTextField~attachTextFieldMaster(masterfields[getField("Last Name",dbfields)
    ]~XPropertySet)
oLNameField~XPropertySet~setPropertyValue("Content","<"||dbfields[getField("Last Name",dbfields)
    ]||">")

oText~XSimpleText~insertString(oText~XTextRange~getEnd(),"Dear ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),oCondField~XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),oLNameField~XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd(),"!",.false)
```

Listing 3.16: 11_FieldExampleFormLetter.rex, second part. Setup of conditional and text body.

A conditional text field, `com.sun.star.text.textfield.ConditionalText`, needs a condition to be set via `"Condition"` property. The `"TrueContent"` and `"FalseContent"` properties contain the textual representation of the field in case the condition evaluates to true or false [Openb, text/textfield/ConditionalText]. Additionally, the property `"IsConditionTrue"` would allow to check the last evaluation result, which is not used in this example. As it is intended to create a gender-specific form of address, the texts are set to `"Miss"` and `"Mister"`. The condition refers to the database field “Sex” and has to contain the database name, the table and the column: `"addresses.address.Sex==1"` (listing 3.16, line 5).

```
/*do the merger*/
/*document already saved?*/
path = xdoc~XModel~getURL()
if path=="" then
do
  path = uno.convertToURL(saveAs)
end
if .bsf.dialog~dialogBox("Document is saved as" path||". OK?", , "warning", "YesNo") == 1 then
EXIT

/*save the document*/
xdoc~XStorable~storeAsURL(path, .UNO~noprops)
```

```
14 oMailMerge = xMCF~createInstanceWithContext("com.sun.star.text.MailMerge",xcc)
   oMailMerge~XPropertySet~setPropertyValue("DataSourceName","addresses")
16 oMailMerge~XPropertySet~setPropertyValue("DocumentURL",xdoc~XModel~getURL())

18 oMailMerge~XPropertySet~setPropertyValue("OutputType",box("SH",bsf.getStaticValue("
       com.sun.star.text.MailMergeType","FILE")))
   oMailMerge~XPropertySet~setPropertyValue("FileNamePrefix", "test_")
20 oMailMerge~XPropertySet~setPropertyValue("SaveAsSingleFile",box("boolean",.true))
   oMailMerge~XPropertySet~setPropertyValue("CommandType",box("I",bsf.getStaticValue("
       com.sun.star.sdb.CommandType","TABLE")))
22 oMailMerge~XPropertySet~setPropertyValue("Command","address")

24 call bsf.import "com.sun.star.beans.NamedValue","NamedValue.class"
   oMailMerge~XJob~execute(bsf.createArray(.NamedValue.class,0))
26
   ::REQUIRES UNO.cls
28
   ::ROUTINE getField
30   USE ARG name, dbfields
     if dbfields~hasItem(name) then
32   do
       RETURN dbfields~index(name)
34   end
     else
36   do
       RETURN -1
38   end
```

Listing 3.17: 11_FieldExampleFormLetter.rex, third part. The merger is finally done.

The final merger in listing 3.17 stores the form letter to the path defined in listing 3.15, line 2 or to the position the document was loaded from. Therefore, the document model's `getURL()` method returns either the document URL or the position it was last saved to, or an empty path in case the document is new. If that is the case, we convert the specified path to a URL via the `uno.convertToURL` method (line 6) [Flat08b]. At any case, the document is stored, the `XStorable` interface of the document model provides `storeAsURL()`.

The merging itself starts with instantiating the `com.sun.star.text.MailMerge` class (line 14), which provides access to the merging functionality. By modifying its parameters, various options can be set. Mandatory properties include the `"DataSourceName"`, which contains the name of the data source to use as well as `"Document"`, which gives the path to the form letter document. `"OutputType"` determines the type of the output, its short value is set by retrieving the constant from `com.sun.star.text.MailMergeType`. Apart from `"FILE"`, which writes the results to a file, `"MAIL"` and `"PRINTER"` are available [Openb, text/MailMergeType]. In this case, `"FILE"` is chosen causing additional properties to be evaluated determining the output URL and the file prefix, both are optional and can also be generated from the source document's properties. For `"MAIL"`, the merger class allows for setting server parameters and passwords or a subject line as well as the possibility to send the document as attachment; for `"PRINTER"`, print options can be defined [Openb, text/MailMerge]. We also set `"SaveAsSingleFile"` to `.true`, the output is therefore written into one single file with a new dataset starting on a new page. Important properties are also `"CommandType"` and `"Command"`. The first describes the

database command type as specified in `com.sun.star.sdb.CommandType`, this constant group offers `"QUERY"` linking it to a query component as defined in data sources, `"COMMAND"` specifies it to be a pure SQL command and `"TABLE"` indicates a table name, which is then processed with an automatically created SQL command [Openb, sdb/CommandType]. In dependence of the command type, `"COMMAND"` then contains the suitable database command. We specify the command to be of type `"TABLE"`, here the command is simply the table name, `"address"`.

The merge statement itself is issued via the **`XJob`** interface, which implements the **`execute`**`()` method. It expects job parameters as a sequence of `com.sun.star.beans.NamedValue` objects [Openb, task/XJob], which are mandatory. Although our example only uses the default parameters, we therefore have to create an empty array of type `NamedValue` (lines 24–25).

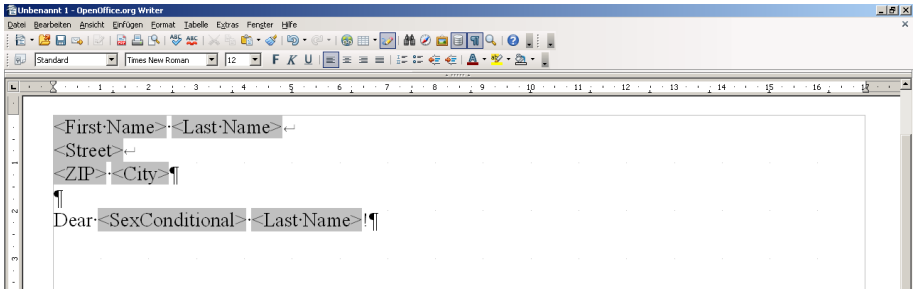


Figure 3.3.: The form letter template as it is used for filling in the data of the “addresses” data source.

3.4. Math Integration

With OpenOffice.org Math, formulas can be created. In most cases, created formulas will be inserted into a text document, which is described in this example.

3.4.1. Example 12 – Insert Formula

As a formula in Writer is an embedded Math object, it first has to be created and then inserted, which is demonstrated by Pitonyak [Pito07, listings 7.68, 7.69, 7.70]. The following script first inserts two formulas, then counts them and shows a possible accession after insertion. It can be run on any Writer document (see figure 3.4).

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
  xcc = xsc~getComponentContext()
6 xMCF = xcc~getServiceManager()
  xMSF = xdoc~XMultiServiceFactory
8
```

```
   /*an example of how to modify formula properties*/
10 fontsize=.array~of(12,18)

12 do i=1 to 2

14 a=RANDOM(2,10)
   b=RANDOM(1,4)
16
   c=a**2+b**2
18 sFormula = a||"^2 +" b||"^2 =" c

20 xController = xdoc~XModel~getCurrentController()
   oViewCursor = xController~XTextViewCursorSupplier~getViewCursor()
22
   oObj = xMSF~createInstance("com.sun.star.text.TextEmbeddedObject")
24 oObj~XPropertySet~setPropertyValue("CLSID","078B7ABA-54FC-457F-8551-6147e776a997")
   oObj~XPropertySet~setPropertyValue("AnchorType",bsf.getStaticValue("
        com.sun.star.text.TextContentAnchorType","AS_CHARACTER"))
26
   xText = oViewCursor~XTextRange~getText()
28 xText~insertTextContent(oViewCursor~XTextRange, oObj~XTextContent, .false)

30 xFormula = oObj~XEmbeddedObjectSupplier~getEmbeddedObject()
   xFormula~XPropertySet~setPropertyValue("Formula",sFormula)
32
   /*set the property*/
34 xFormula~XPropertySet~setPropertyValue("BaseFontHeight",box("SH",fontsize[i]))
   end
36
   /*all adaptable properties can be listed using the following code lines*/
38 --props = uno.getProperties(xFormula)~makeArray(" ")~toString
   --xText~XSimpleText~insertString(oViewCursor~XTextRange,props,.false)
40
   /*let's count the formulas in the document*/ --
42 s=""
   oEmbed = xdoc~XTextEmbeddedObjectsSupplier~getEmbeddedObjects()
44
   do i=1 to oEmbed~XIndexAccess~getCount()
46   oCurObj = oEmbed~XIndexAccess~getByIndex(i-1)
     if oCurObj~XServiceInfo~supportsService("com.sun.star.text.TextEmbeddedObject") then
48   do
       if oCurObj~XPropertySet~getPropertyValue("CLSID")=="078B7ABA-54FC-457F-8551-6147e776a997"
            then
50     do
         xCurFormula = oCurObj~XEmbeddedObjectSupplier~getEmbeddedObject()
52       s = s xCurFormula~XPropertySet~getPropertyValue("Formula") .ENDOFLINE
       end
54   end
   end
56
   .bsf.dialog~messageBox(s)
58 ::requires UNO.cls
```

Listing 3.18: 12_InsertFormula.rex. Inserts a OpenOffice.org Math formula into a Writer document.

The formula creation and insertion in listing 3.18 is processed within an iteration to build two very similar formulas, the numbers are generated randomly out of a predefined range (lines 14–15). Additionally one formula property is examplarily modified, the basic font size, which is changed from 12 to 18 for the second derivative of the formula $a^2 + b^2 = c$. OpenOffice.org Math has a broad range of object properties, all available can be listed by uncommenting lines 37–39. To write correct results, ooRexx calculates the equation result, which is then written in the next step according to the normal OpenOffice.org Math syntax (line 18).

Then an object of type `com.sun.star.text.TextEmbeddedObject` is instantiated, it is of type base frame (compare [Sun07, p. 602]) and contains a document type different from the current, e.g. a chart or OLE (Object Linking and Embedding)-object. The formula is a OpenOffice.org Math document inserted into Writer. The `TextEmbeddedObject` needs the `"CLSID "` (Class Identifier) parameter to be set, it is a globally unique identifier for the embedded document type, a COM class object. The Math CLSID is "078B7ABA-54FC-457F-8551-6147e776a997" (line 24). The `"AnchorType"` parameter is responsible for the embedded object's behaviour within the text, it is a property of the **`XTextContent`** interface which is supported by all objects insertable into text [Sun07, p. 559]. Its value is set according to `com.sun.star.text.TextContentAnchorType`, which allows an anchoring instead of a character (`"AS_CHARACTER"`, as in listing 3.18), to a character, a frame, a page or a paragraph [Openb, text/TextContentAnchorType] (ooRexx examples in [Gmei08, Scho09, Schm07]).

Afterwards, the formula object is inserted via `insertTextContent()`. Still, the object itself does not show our formulas, we first have to assign it to the property `"Formula"` of the formula object itself, which can be retrieved via the **`XEmbeddedObjectSupplier`** and the `getEmbeddedObject()` method of the base frame object. This is only possible after the insertion into the text, otherwise the method returns `.NIL`. To demonstrate one formula's appearance modification, we reset the `"BaseFontHeight"`, a property from which all text sizes, e.g. the superscript size, are derived from (line 34).

In the second part of listing 3.18 (lines 41–57), all formulas of a document are retrieved. Similarly, it is possible to access all embedded objects or other embedded object types. The Writer model's **`XTextEmbeddedObjectsSupplier`** interface provides `getEmbeddedObjects()`, which returns an enumeration of embedded objects accessible via an **`XIndexAccess`** interface.

Then a two-step process determines whether the embedded object is a Math document. First, it is ensured that the `TextEmbeddedObject` service is supported (line 47), which in principle all objects returned by `getEmbeddedObjects()` should [Openb, text/XTextEmbeddedObjectsSupplier]. Pitonyak also performs this redundant mechanism [Pito07, listing 7.70]. Second, a `"CLSID"` property comparison identifies Math embedded objects. Then the `"Formula"` property is extracted after accessing the embedded object. The formulas are then displayed.

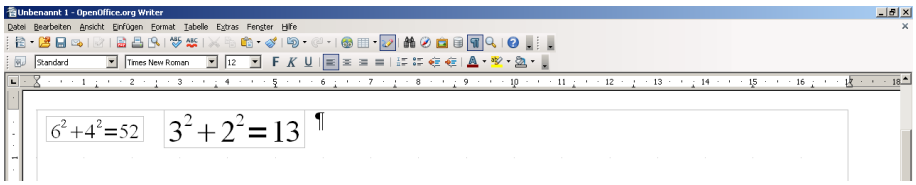


Figure 3.4.: Two formulas are inserted into Writer.

4. Selected Pitonyak OpenOffice.org Calc Examples

As for OpenOffice.org Writer, many examples are available demonstrating the get-together of ooRexx and Calc. Here some additional methods on how to operate and set formulas on whole cell ranges are provided.

4.1. Supporting Operations on Cell Ranges

Cell ranges are rectangular subparts of a spreadsheet's cells, which offer some functionality referring to the block as a whole. This section introduces various methods of manipulating cell ranges.

4.1.1. Example 13 – Sort Cell Ranges

When scripting OpenOffice.org Calc, sorting specified cell ranges might be interesting. The GUI sorting methods can be accessed via "Data–Sort" from the menu and allow to specify various criteria, which can also be accessed by script. The following example 4.1 sorts a Calc sheet taking into account the first two columns including a header row, the first column is sorted ascending, if the second criteria has to be applied, it is configured to sort descending. With this mechanism of Calc, whole cell rows are moved. The script is adapted to ooRexx from [Pito07, listing 6.28], another example of a sort implementation can be found in [Sun07, p. 695] and an ooRexx variant in [Aham05]. The script can be executed on in principle any Calc document, a useful preparation script can be found in appendix A.1, listing A.3.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

```

```
/*currently selected sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()
xCellRange = oSheet~XSheetCellRange~XCellRange

call bsf.import "com.sun.star.table.TableSortField","TableSortField.class"
aSortFields = bsf.createArray(.TableSortField.class,2)
aSortFields[1]=.bsf~new("com.sun.star.table.TableSortField")
aSortFields[1]~bsf.setFieldValue("Field",0)
aSortFields[1]~bsf.setFieldValue("IsAscending","TRUE")
aSortFields[1]~bsf.setFieldValue("IsCaseSensitive","FALSE")

aSortFields[2]=.bsf~new("com.sun.star.table.TableSortField")
aSortFields[2]~bsf.setFieldValue("Field",1)
aSortFields[2]~bsf.setFieldValue("IsAscending","FALSE")
aSortFields[2]~bsf.setFieldValue("IsCaseSensitive","FALSE")

aSortDesc = bsf.createArray(.UNO~PROPERTYVALUE,2)
aSortDesc[1] = uno.createProperty("SortFields",aSortFields)
aSortDesc[2] = uno.createProperty("ContainsHeader",box("boolean",.true))

xCellRange~XSortable~sort(aSortDesc) --

::requires UNO.cls
```

Listing 4.1: 13_SortRange.rex. Sorts a Calc sheet content's according to first two columns.

Before sorting the sheet's data, we have to obtain a cell range to sort. In our case, we select the currently active sheet. The cell range is contained in `xCellRange` and could also be set to a subrange.

For sorting, a sort descriptor is used, a `com.sun.star.sheet.SheetSortDescriptor2` service which acts within one sheet. The most important is derived from the included service `com.sun.star.table.TableSortDescriptor2`, the `"SortFields"` property. It defines both the columns which are taken into account and their sort properties. Additonally, some options allow to exactly define the sorting, e.g. the property `"ContainsHeader"` decides whether the first row is ignored, or with `"BindFormatToContent"` the formatting of the cells is bound to their contents when moved [Openb, sheet/SheetSortDescriptor2]. Apart from retrieving the search descriptor from the `createSortDescriptor()` method of the **`XSortable`** interface of a cell range, it can also be created from the scratch, as it only consists of its various properties. The retrieval method might contain sort settings saved within the document, therefore we build the descriptor ourselves and only define the sort fields (lines 25–27). For properties not explicitly set, the default values are used [Sun07, p. 694].

Columns to include are defined with `com.sun.star.table.TableSortField`, which replaces the deprecated `com.sun.star.util.SortField` [Openb, util/SortField]. In principle, both structures are similar, the `"Field"` property indicates the column they are applied to starting with 0, `"FieldType"` would allow to set the data type of the column. If not set, an algorithm tries to detect the column type automatically. The `"SortAscending"` property of the `SortField` is called `"IsAscending"` in `TableSortField`. Apart from these, the latter offers additional properties [Openb, util/SortField and table/TableSortField], e.g. a case sensitivity

flag as well as the integration of a collator algorithm, which influences the sort criteria according to a certain locale [Sun07, p. 694]. These new properties are not fully supported by OpenOffice 3.0, they can't be set differently for each field without giving undefined results [Openb, sheet/SheetSortDescriptor2].

In line 13, the `TableSortField` class is registered with BSF, then the array is built and filled. In the GUI version, a maximum of three fields can be assigned to the sort descriptor. Although more than three sort fields can be added to a descriptor via script, Calc ignores those exceeding the third. Thus, scripting does not solve the problem of sorting more than three columns in Calc (compare [Open03]).

The sorting itself is triggered via the `sort()` method of the **`XSortable`** interface, which expects a property sequence from the sort descriptor as parameter (line 29).

4.1.2. Example 14 – Conditional Formatting and Cell Validation

In some cases, conditional cell formatting might be interesting to obtain a better overview. Additionally, a cell validation service allows for defining exactly which values can be inserted into a cell. In OpenOffice.org Calc, both features are implemented as sheet cell range and cell properties, the other properties are position and size. Position is the upper-left most cell location, the size property contains the size of the cells [Pito04, p. 334].

Before starting with a validation or with a conditional formatting, some prerequirements have to be met, which are checked in listing 4.2. A cell range has to be selected, and certain styles used to indicate condition matching have to be defined. Our example uses the styles "red" and "green", a creation script is available in listing A.5. The method of querying the presence of styles by name is taken from Frysak [Frys08] (lines 12–25), the specific style type is `"CellStyles"`. The script exits if an error concerning styles occurs. It can be executed from within a Calc document, a data preparation script is available in appendix A.1, listing A.3.

The cell range is defined in the variable `sCellRange` in line 10. Again, the method of selecting the active spreadsheet and the range is inspired by Frysak [Frys08] and saved in `oRange`.

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
  xcc = xsc~getComponentContext()
6 xMCF = xcc~getServiceManager()
  xMSF = xdoc~XMultiServiceFactory
8
  /*define cell range where the conditional formatting is applied to*/
10 sCellRange="B1:D10"

12 /*we need the styles "red" and "green" for conditional formatting - check whether they are
       present*/
```

```
aStyles=.array~of("red","green")
xStyleFamilies = xdoc~XStyleFamiliesSupplier~getStyleFamilies()

/*in OOo.org Calc there are 2 style families: CellStyles and PageStyles*/
xStyleFamily = xStyleFamilies~getByName("CellStyles")

do i=1 to aStyles~items
  if xStyleFamily~XNameAccess~hasByName(aStyles[i])==0 then
  do
    .bsf.dialog~messageBox("Style" aStyles[i] "not present. Exiting")
    EXIT
  end
end

/*currently selected sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()
oRange = oSheet~XSheetCellRange~getCellRangeByName(sCellRange)
```

Listing 4.2: 14_ConditionalValidatedFormatting.rex, first part. Ensures that required styles are present.

The cell validation service of OpenOffice.org Calc does not only prevent invalid data from being entered into a cell, it additionally offers dialogs and popups to inform the user about the limitation, as Pitonyak demonstrates [Pito04, p. 336]. The second part of our example referring to this Pitonyak script, listing 4.3, sets a number range between 1.4 and 10 to the cells selected in listing 4.2 and implements various user dialogs.

```
/**VALIDATION**/
/*get the validation object
 *alternatively we can create a PropertySet*/
oValidation = oRange~XPropertySet~getPropertyValue("Validation")

/*configure the validation range*/
oValidationProps = oValidation~XPropertySet
oValidationProps~setPropertyValue("Type",bsf.getStaticValue("com.sun.star.sheet.ValidationType","
    DECIMAL"))
oValidationProps~setPropertyValue("ErrorMessage","Please enter a number between 1.4 and 10.")
oValidationProps~setPropertyValue("ErrorTitle","Wrong Input Range.")
oValidationProps~setPropertyValue("ErrorAlertStyle",bsf.getStaticValue("
    com.sun.star.sheet.ValidationAlertStyle","STOP"))
oValidationProps~setPropertyValue("ShowErrorMessage",box("boolean",.true))
oValidationProps~setPropertyValue("InputMessage","Number 1.4 to 10.")
oValidationProps~setPropertyValue("InputTitle","Hint")
oValidationProps~setPropertyValue("ShowInputMessage",box("boolean",.true))

oValidation~XSheetCondition~setOperator(bsf.getStaticValue("com.sun.star.sheet.ConditionOperator"
    ,"BETWEEN"))

/*decimal numbers have to be entered using a point, they are converted to the correct format in
    OOo.org*/
oValidation~XSheetCondition~setFormula1(1.4)
oValidation~XSheetCondition~setFormula2(10)

oRange~XPropertySet~setPropertyValue("Validation",oValidation)
```

Listing 4.3: 14_ConditionalValidatedFormatting.rex, second part. A cell validation service is set.

The validation process is included in the `com.sun.star.sheet.TableValidation` service and controlled via various enumerations setting the properties of that service. As for the search descriptor (compare listing 4.1), the validation property can be built up either from the scratch or modified. In that case, the already existing `TableValidation` set as `"Validation"` property is changed (lines 4–15) [Pito04, p. 334]. Its `"Type"` property can be set to values contained

in the com.sun.star.sheet.ValidationType enumeration, which includes fields from "ANY" for any type of content to "DATE", which only accepts dates as input. A list can be found in [Pito04, p. 334] or [Openb, sheet/ValidationType]. The "ErrorMessage" property can be set to a string displayed when the user tries to enter invalid content into a cell, "ErrorTitle" allows to specify an appropriate title and the "ErrorAlertStyle" controls the popup type, e.g. "STOP" or "WARNING" [Openb, sheet/ValidationAlertStyle]. This property also influences Calc's behaviour, the first rejects the input, while the second only gives a warning and asks the user whether the invalid content should be accepted. Similarly, "InputMessage" and "InputTitle" define an information shown when the cursor enters the cell. In both cases, a flag controlling the display of the messages can be set ("ShowErrorMessage" and "ShowInputMessage").

The actual validation condition is then defined via the **XSheetCondition** interface of the validation service. Its methods allows to define (setX()) and retrieve (getX()) an operator of the validation condition as well as its first and, if necessary, second formula value. The operators (setOperator()) are defined in com.sun.star.sheet.ConditionOperator (see [Pito04, p. 335] or [Openb, sheet/ConditionOperator] for valid values), the "BETWEEN" shown in this example and "NOT_BETWEEN" operators need both formulas to be set, for the others the setFormula1() method is sufficient. This is set to 1.4 and the setFormula2() method to 10. Resultingly, inputs between 1.4 and 10 are valid after the validation property has been set (line 23).

Conditional formatting sets the cell style in dependence of various criteria to predefined styles. Pitonyak shows an example [Pito04, p. 336], which is transcribed to ooRexx in listing 4.4. Here, the cell style is set to "red" if the content is less than 5 and to "green" if the content is exactly equal to 10. The styles color the cell background with the respective color.

```
/**CONDITIONAL FORMATTING**/
/*set a conditional formatting to the cells selected above*/

/*get the conditional formatting object*/
oCondFormat = oRange~XPropertySet~getPropertyValue("ConditionalFormat")

/*build up the conditions that have to apply*/
oFirstCondition = uno.createArray(.UNO~PROPERTYVALUE,3)
oFirstCondition[1] = uno.createProperty("Operator",bsf.getStaticValue("
     com.sun.star.sheet.ConditionOperator","LESS"))
oFirstCondition[2] = uno.createProperty("Formula1",5)
oFirstCondition[3] = uno.createProperty("StyleName",aStyles[1])

oCondFormat~XSheetConditionalEntries~addNew(oFirstCondition)

oSecondCondition = uno.createArray(.UNO~PROPERTYVALUE,3)
oSecondCondition[1] = uno.createProperty("Operator",bsf.getStaticValue("
     com.sun.star.sheet.ConditionOperator","EQUAL"))
oSecondCondition[2] = uno.createProperty("Formula1",10)
oSecondCondition[3] = uno.createProperty("StyleName",aStyles[2])

oCondFormat~XSheetConditionalEntries~addNew(oSecondCondition)

oRange~XPropertySet~setPropertyValue("ConditionalFormat",oCondFormat)
```

```
24 ::requires UNO.cls
```

Listing 4.4: 14_ConditionalValidatedFormatting.rex, third part. A conditional service is set.

The conditional formatting is set up similarly to the validation service in listing 4.4. At first, we retrieve the `"ConditionalFormat"` property value of the range, which is of type `com.sun.star.sheet.TableConditionalFormat`. It represents a collection of conditional formattings assigned to a cell or cell range [Openb, sheet/TableConditionalFormat] and provides both an **`XIndexAccess`** and an **`XNameAccess`** interface to access the conditions set, the name is given automatically and equals `"Entry0"` for the first and `"Entry1"` for the second condition.

The conditions themselves are property sequences, supported properties are `"Operator"` specifying the operator as for the validation, `"Formula1"` and `"Formula2"`, again with the same meaning as before with the validation service, and `"StyleName"` pointing to the style the cell is assigned if the condition applies. The `"SourcePosition"` property allows for defining a source position for relative cell formula references, it is not used in the script [Openb, sheet/XSheetConditionalEntries]. If more than one condition defined for a cell applies, the first valid is taken [Pito04, p. 336].

In the macro example 4.4, two conditions are defined, the first having a `"LESS"` operator and 5 as formula value, so that every number less than 5 validates to true. For the second, this is the case if the entry is equal to 10, as the operator is set to `"EQUAL"`. Each condition has to be added to the `TableConditionalFormat` with the `addNew()` method of the **`XSheetConditionalEntries`** interface, and the whole property has to be set for the cell range.

4.1.3. Example 15 – Cell Queries

OpenOffice.org's cell ranges and cells support a feature which manages to find cells having specific properties, e.g. a certain type of value like a string or a date, but also certain styles or objects assigned to them. Pitonyak presents some of these cell queries in [Pito04, p. 338–341], the following script reproduces a simple cell query using ooRexx. It prints out all non-empty cells of a specified range on any Calc document, e.g. after executing listing A.3, appendix A.1.

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
  xcc = xsc~getComponentContext()
6 xMCF = xcc~getServiceManager()
  xMSF = xdoc~XMultiServiceFactory
8
  sRange = "A1:F5"
```

```
10
   /*select procedure as in Frysak2008*/
12 /*currently selected sheet*/
   oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()
14 oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

16 /*use of the CellAddressConversion service to get human readable cell addresses*/
   oConversion = xMSF~createInstance("com.sun.star.table.CellAddressConversion")
18 oFlags = .array~of( -
     bsf.getStaticValue("com.sun.star.sheet.CellFlags","VALUE"), -
20   bsf.getStaticValue("com.sun.star.sheet.CellFlags","FORMULA"), -
     bsf.getStaticValue("com.sun.star.sheet.CellFlags","STRING"), -
22   bsf.getStaticValue("com.sun.star.sheet.CellFlags","DATETIME") -
   )
24
   /*sum up*/
26 sSum=0
   do i=1 to oFlags~items
28 sSum = sSum + oFlags~at(i)
   end
30
   oRanges = oRange~XCellRangesQuery~queryContentCells(sSum)
32
   oAddrs = oRanges~getCells()~XEnumerationAccess~createEnumeration()
34 s="Nonempty cells of" sRange .ENDOFLINE
   do while oAddrs~hasMoreElements
36   oCell = oAddrs~nextElement
     /*convert to human readable format*/
38   oCellAddress = oCell~XCellAddressable~getCellAddress()
     oConversion~XPropertySet~setPropertyValue("Address",oCellAddress)
40   s = s oConversion~XPropertySet~getPropertyValue("UserInterfaceRepresentation") .ENDOFLINE
   end
42
   .bsf.dialog~messageBox(s)
44
   ::requires UNO.cls
```

Listing 4.5: 15_CellQueries.rex. Non-empty cells of a specified range are retrieved.

At first, listing 4.5 defines a cell range to deal with (line 9), in that case `"A1:F5"`, then the range is retrieved from the current sheet. The bit flags for the search are defined as Frysak examplified [Frys08], they can be retrieved from the `com.sun.star.sheet.CellFlags` constant group. In order to get all non-empty cells, we define the search to contain the `"VALUE"` flag, which selects constant numeric values, the `"DATETIME"` flag to get all dates and times, the `"STRING"` includes all text string cells and the `"FORMULA"` cells with formulas [Openb, sheet/CellFlags; also for other flags]. The summed up constant values are the parameter of the **`queryContentCells`**`()` method supplied by the **`XCellRangesQuery`** interface of the cell range. Resultingly, the query is based on a cell's content, but the interface offers additional methods like querying all visible cells, all empty cells or row and column differences, which has already been demonstrated by Burger [Burg06]. All interface methods return an **`XSheetCellRanges`** interface, which supplies an **`XIndexAccess`** interface to address the contained cell ranges and additionally provides some specialized methods for cell and range retrieval, e.g. **`getCells`**`()` returning all cells, or **`getRangeAddresses`**`()` for the included cell ranges [Openb, sheet/XSheetCellRanges]. Pitonyak uses the **`getRangeAddresses`**`()` method, which requires a more complicated processing which is not necessary if it is only intended to retrieve non-empty cells [Pito04, p. 338].

The ooRexx transcript utilizing Pitonyak's variant can be found in appendix A, listing A.8, it substitutes lines 34–41.

The better suited `getCells()` method provides an **`XEnumerationAccess`** interface providing the `createEnumeration()` method. An iteration is done via `hasMoreElements()` signalling whether the collection's end has been reached and `nextElement()` giving the next element, an **`XCell`** implementing **`XCellAddressable`**, which allows to get the cell's `CellAddress` value representing a structure with a numeric value for the cell's absolute position (row, column).

Nonetheless, this is different from the normally used alphanumeric column and row identifiers, so a conversion to the "human readable" format is done. We therefore instantiate the `com.sun.star.table.CellAddressConversion` service (line 17). This by May 2009 API-undocumented service has been present since OpenOffice.org 1.1.1 and is described in [Pito04, p. 328]. The cell address is set via its `"Address"` property (line 39), it then offers various representations including `"PersistentRepresentation"` or `"UserInterfaceRepresentation"`. The first always adds the sheet name, the second only if the cell is not on the currently active. The script utilizes the last (line 40).

4.1.4. Example 16 – Cells with same Formatting

OpenOffice.org offers a feature which returns all cells of a cell range having the same formatting, which is implemented in two variants with different return types. Examples can be found in the Pitonyak book [Pito04, p. 350] as well as in the Developer's Guide [Sun07, p. 675–678]. An ooRexx version is provided in listing 4.6, which can be invoked on any Calc sheet. An examplary data generation script can be found in listing A.3, appendix A.1.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*select procedure as in Frysak2008*/
/*currently selected sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()
/*obtain current sheet name and number*/
iSheetNumber = oSheet~XCellRangeAddressable~getRangeAddress()~bsf.getFieldValue("Sheet")
sSheetName = oSheet~XNamed~getName()

/*for cell address conversion*/
oConversion = xMSF~createInstance("com.sun.star.table.CellAddressConversion")

/*getCellFormatRanges()*/
s = "*** getCellFormatRanges() ***" .ENDOFLINE
oRanges = oSheet~XCellFormatRangesSupplier~getCellFormatRanges()

do i = 1 to oRanges~XIndexAccess~getCount()
  oRange = oRanges~XIndexAccess~getByIndex(i-1)
```

```
    oAddr = oRange~XCellRangeAddressable~getRangeAddress()
26  oStartAddr = .bsf~new("com.sun.star.table.CellAddress")
    oStartAddr~bsf.setFieldValue("Sheet",iSheetNumber)
28  oStartAddr~bsf.setFieldValue("Column",oAddr~bsf.getFieldValue("StartColumn"))
    oStartAddr~bsf.setFieldValue("Row",oAddr~bsf.getFieldValue("StartRow"))
30  oEndAddr = .bsf~new("com.sun.star.table.CellAddress")
    oEndAddr~bsf.setFieldValue("Sheet",iSheetNumber)
32  oEndAddr~bsf.setFieldValue("Column",oAddr~bsf.getFieldValue("EndColumn"))
    oEndAddr~bsf.setFieldValue("Row",oAddr~bsf.getFieldValue("EndRow"))
34
    s = s i||":" sSheetName||"."
36  /*convert start address*/
    oConversion~XPropertySet~setPropertyValue("Address",oStartAddr)
38  s= s||oConversion~XPropertySet~getPropertyValue("UserInterfaceRepresentation")
    /*convert end address*/
40  oConversion~XPropertySet~setPropertyValue("Address",oEndAddr)
    s= s ||":"|| oConversion~XPropertySet~getPropertyValue("UserInterfaceRepresentation")
        .ENDOFLINE
42 end
   s= s .ENDOFLINE
44
   /*getUniqueCellFormatRanges()*/
46 s = s "*** getUniqueCellFormatRanges() ***" .ENDOFLINE
   oRanges = oSheet~XUniqueCellFormatRangesSupplier~getUniqueCellFormatRanges()
48
   do i=1 to oRanges~XIndexAccess~getCount()
50   oRangeContainer = oRanges~XIndexAccess~getByIndex(i-1)
     s = s i||":" oRangeContainer~XSheetCellRanges~getRangeAddressesAsString() .ENDOFLINE
52 end

54 .bsf.dialog~messageBox(s)

56 ::requires UNO.cls
```

Listing 4.6: 16_CellsSameFormatting.rex. Returns cells with similar formatting.

Listing 4.6 demonstrates how to obtain the current sheet's number and its name, which is later used to show the user the retrieved cell ranges. The number can be gotten via the **XCellRangeAddressable** method interface of the range and the **getRangeAddress()**, which returns a `CellAddress` structure including the sheet, column and row in numerical representation [Openb, sheet/XCellRangeAddressable]. The name is given back by the **getName()** method of the range's **XNamed** interface [Openb, container/XNamed].

The macro uses both cell range retrieving methods. The first cell range variant called is **getCellFormatRanges()** available via the **XCellFormatRangesSupplier** interface, the second **getUniqueCellFormatRanges()** supplied via **XUniqueCellFormatRangesSupplier**. In principle, both are quite similar. The first returns one rectangular range per formatting allowing an **XEnumerationAccess** and and **XIndexAccess** and splits non-rectangular formatting areas into n rectangular fields, while the second combines all rectangular ranges of same type into one container of type `com.sun.star.sheet.SheetCellRanges` [Pito04, Sun07].

In line 21, the **getCellFormatRanges()** method is called. The returned ranges are then listed on a dialog box with their left upper and right lower edges describing them. We therefore iterate over them via **XIndexAccess** and retrieve the range address of each range, which is saved in a `com.sun.star.table.CellRangeAddress` object coding the sheet and the starting and ending

cells. Two new cell address objects are instantiated and filled with these references. This is necessary as we want to convert the numeric representation into the human readable using the undocumented conversion feature (see listing 4.5) and present the range in the representation "SheetName.StartCell:SheetName.EndCell".

Similarly, we retrieve `getUniqueCellFormatRanges()` (line 47). We iterate over the container via the `XIndexAccess` interface. The container supports `XSheetCellRanges`, which implements the `getRangeAddressesAsString()` method. This returns the already formatted cell ranges, which are put onto the user dialog.

The listing finally demonstrates that both methods in principle return the same ranges, but the `getUniqueCellFormatRanges()` groups them further according to their formatting.

4.2. Formula Operations on Cell Ranges

Formulas can't only be defined on one cell only, some types have to be assigned to a certain cell range, e.g. matrix formulae, which are discussed in the following examples.

4.2.1. Example 17 – Automatic Filling

Automatic filling methods are a comfortable way of entering consecutive data into cells. Apart from the automatic series generic from extending a selected cell to cells nearby with the mouse, OpenOffice.org offers more advanced methods in the GUI version (Edit–Fill...), which are also accessible via the API. Pitonyak presents some short examples dealing with automatic filling [Pito04, p. 345–347], script 4.7 gives an overview over various filling methods in ooRexx: a simple fill with consecutive numbers (already demonstrated in [Aham05]), a constant fill, a date fill and a geometric fill. The example should be invoked on an empty Calc sheet.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*select procedure as in Frysak2008*/
/*currently selected sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

/**SIMPLE FILL**/
call uno.setCell oSheet,"A1","simple increment every 2nd cell"
sRange = "B1:L1"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

/*set start value*/
```

```
call uno.setCell oSheet,"B1",8

/*simple increment fill in every 2nd cell*/
oRange~XCellSeries~fillAuto(bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_RIGHT"),2)

/**CONSTANT FILL**/
call uno.setCell oSheet,"A2","simple constant each cell from left"
sRange = "B2:L2"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

/*set start value*/
call uno.setCell oSheet,"L2",13

/*simple constant fill in each cell from left*/
oRange~XCellSeries~fillSeries(-
  bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_LEFT"),-
  bsf.getStaticValue("com.sun.star.sheet.FillMode","SIMPLE"),-
  bsf.getStaticValue("com.sun.star.sheet.FillDateMode","FILL_DATE_YEAR"),-
  1,10000)

/**INCREMENT OF MONTH**/
call uno.setCell oSheet,"A3","increment of months"
sRange = "B3:L3"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

/*set start value*/
date = DATE()
call uno.setCell oSheet,"B3", date

/*simple constant fill in each cell from left*/
oRange~XCellSeries~fillSeries(-
  bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_RIGHT"),-
  bsf.getStaticValue("com.sun.star.sheet.FillMode","DATE"),-
  bsf.getStaticValue("com.sun.star.sheet.FillDateMode","FILL_DATE_MONTH"),-
  3,50000)

/*we also have to define the cell format as DATE*/
/*retrieve the standard DATE format*/
/*we have to define the locale*/
oLocale = .bsf~new("com.sun.star.lang.Locale")
oLocale~bsf.setFieldValue("Country","USA")
oLocale~bsf.setFieldValue("Language","en")
oNumberFormats = xdoc~XNumberFormatsSupplier~getNumberFormats()
iFormatIndex = oNumberFormats~XNumberFormatTypes~getStandardFormat(bsf.getStaticValue("
     com.sun.star.util.NumberFormat","DATE"),oLocale)
oRange~XPropertySet~setPropertyValue("NumberFormat",box("i",iFormatIndex))

/**GEOMETRIC SERIES FROM TOP**/
call uno.setCell oSheet,"A4","geometric series from top to bottom"
sRange = "A5:A40"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

/*set start value*/
call uno.setCell oSheet,"A5", 1

/*simple constant fill in each cell from left*/
oRange~XCellSeries~fillSeries(-
  bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_BOTTOM"),-
  bsf.getStaticValue("com.sun.star.sheet.FillMode","GROWTH"),-
  bsf.getStaticValue("com.sun.star.sheet.FillDateMode","FILL_DATE_MONTH"),-
  5,100000)

::requires UNO.cls
```

Listing 4.7: 17_AutomaticFilling.rex. Demonstrates various filling methods.

In general, the automatic filling is handled via the **`XCellSeries`** interface of a cell range. This interface provides two methods. `fillAuto()` gives quick access, expecting only the direction

as coded in the enumeration com.sun.star.sheet.FillDirection and a long i indicating which i^{th} cell has to be filled. The fillSeries() method provides additional options [Openb, sheet/XCellSeries]. FillDirection provides options to the left, right, bottom and top [Openb, sheet/FillDirection].

The first example in the script (lines 13–22) uses the simple variant to define a whole number series, which is increased by one and filled into every second cell. After retrieving the cell range, we set the start value (cell B1) to 8 and then call the fillAuto() method. It always references the first cell in the range (cells B1–L1), thus 8 [Openb, sheet/XCellSeries]. The parameters are then set to "TO_RIGHT", the count to 2 (line 22).

In the second part, the more advanced fillSeries() method is called to fill a series of cells with a constant (lines 24–37). As for the simple variant, a range is retrieved and the constant value set to the first cell. It needs five parameters, apart from the fill direction it supports the fill mode options (from com.sun.star.sheet.FillMode) as well as the fill data mode options (from com.sun.star.sheet.FillDateMode), a step parameter to specify the value of the consecutive increase and an end value indicating at which number the filling should stop [Openb, sheet/XCellSeries]. These are set from line 33–37. The fill mode is set to "SIMPLE" indicating a constant, the other parameters nonetheless have to be provided. Fill data mode only has a meaning when dealing with dates (compare [Openb, sheet/FillDateMode]), the step parameter is also of no use when defining a constant filling.

For the third filling, a date type (FillMode "DATE") is taken (lines 39–63). The first cell is set to the date retrieved by the ooRexx function DATE(). In this case, the FillDateMode is of importance, as it influences which part of the date is subject to modification, we set it to "FILL_DATE_MONTH". In this case, the step parameter is referring to the month of the date, while the day and the year are unchanged [Openb, sheet/FillDateMode; also for other options]. We set the step to three, so two following dates are a quarter of a year apart. Experiments show that if the start day is set e.g. to the 31$^{\text{st}}$ which is not available for all months, the series is modified to the closest day available, so e.g. for June to the 30$^{\text{th}}$.

Up to now, the cell formatting has not been changed to "date" format, it shows the numerical representation. We therefore retrieve the default date value for the USA (lines 55–63). A new com.sun.star.lang.Locale is instantiated to define the country and the language, then the availabe number formats are returned (see listing 3.10 for details) and queried for a default format, in this case of the USA, via the XNumberFormatTypes interface specified by the com.sun.star.util.NumberFormats container. This interface allows the retrieval of format indices of predefined types, getStandardFormat() expects both a locale and a special number format constant defined in the com.sun.star.util.NumberFormat [Sun07, p. 519] constant

group, in our case `"DATE"` [Openb, util/NumberFormat]. The format is then applied to the range via the property `"NumberFormat"`.

The last example in lines 65–78 fills a cell series with a geometric series to the bottom. Therefore, the `FillDirection` is set to `"TO_BOTTOM"` and the `FillMode` to `"GROWTH"`. The step is 5, so cell $n + 1 = 5 * n$. Although the cell range has been defined from A5 to A40, the last value filled is 78125 in cell A12. This is caused by the step-in of end value threshold, which is set to 100000 in this case (line 78).

4.2.2. Example 18 – Array Formulas

Array formulas need more than one cell to be defined correctly. The script sets two examples, both are taken from Pitonyak [Pito04, p. 348]. The macro should be invoked on an empty Calc sheet.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*select procedure as in Frysak2008*/
/*currently selected sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

/**SIMPLE ARRAY FORMULA EXAMPLE**/
/*first create some input data*/
/*see listing 23 "AutomaticFills.rex" for details*/
sRange = "A2:B11"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)
call uno.setCell oSheet,"A1","a"
call uno.setCell oSheet,"B1","b"
call uno.setCell oSheet,"A2",1
call uno.setCell oSheet,"B2",10
oRange~XCellSeries~fillAuto(bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_BOTTOM"),1)


/*now set the array formula*/
call uno.setCell oSheet,"C1","a*b"
oResultRange = oSheet~XSheetCellRange~getCellRangeByName("C2:C11")
oResultRange~XArrayFormulaRange~setArrayFormula("=A2:A11*B2:B11")

/**EXAMPLE: FREQUENCY ARRAY FUNCTION**/
sRange = "A2:C11"
oRange = oSheet~XSheetCellRange~getCellRangeByName(sRange)

call uno.setCell oSheet,"A15","Frequencies"
call uno.setCell oSheet,"A16",5
call uno.setCell oSheet,"A17",15
call uno.setCell oSheet,"A18",25
call uno.setCell oSheet,"A19","larger"

/*now set the array formula*/
call uno.setCell oSheet,"C1","a*b"
oResultRange = oSheet~XSheetCellRange~getCellRangeByName("B16:B19")
oResultRange~XArrayFormulaRange~setArrayFormula("=FREQUENCY(A2:C11;A16:A18)")
```

```
44 ::requires UNO.cls
```

Listing 4.8: 18_ArrayFormulas.rex. This script gives two simple array formula examples.

Before setting an array formula, some data have to be inserted. The script in listing 4.8 uses the automatic filling procedures introduced in listing 4.7 to produce two columns containing consecutive numbers bottomwards (lines 14–22). Afterwards, the array formula can be set, it should multiply the first two columns in the third. At first place, the target cell range C2–C11 is retrieved, in the second step the array formula is assigned. That can be done via `XArrayFormulaRange` implemented in `XCellRange` and `setArrayFormula()` method, which in this case is `"=A2:A11*B2:B11"` (line 27). Apart from self-defined array formulas, OpenOffice.org offers array formulas within its predefined functions, e.g. the FREQUENCY function. The second array formula thus defines the whole range covered in the first example as an input and defines three classes of frequency in the cells A16–A18; the result than specifies the number of cell value occurrances less or equal 5, between 5 and 15, 15 and 25 and larger. The result range thus compasses four cells, as additionally the "larger" option is included (B15–B19)! Then the array formula can be assigned: `"=FREQUENCY(A2:C11;A16:A18)"` (line 42). The first parameter points to the input cells, the second to the classes to take into account. It is important to notice that via `setArrayFormula()` all functions are set in English, similar to `setFormula()` (see chapter 2.3.1). The current locale which is displayed in the GUI can be accessed via the `"FormulaLocal"` property of the cell.

4.2.3. Example 19 – Multiple Operations

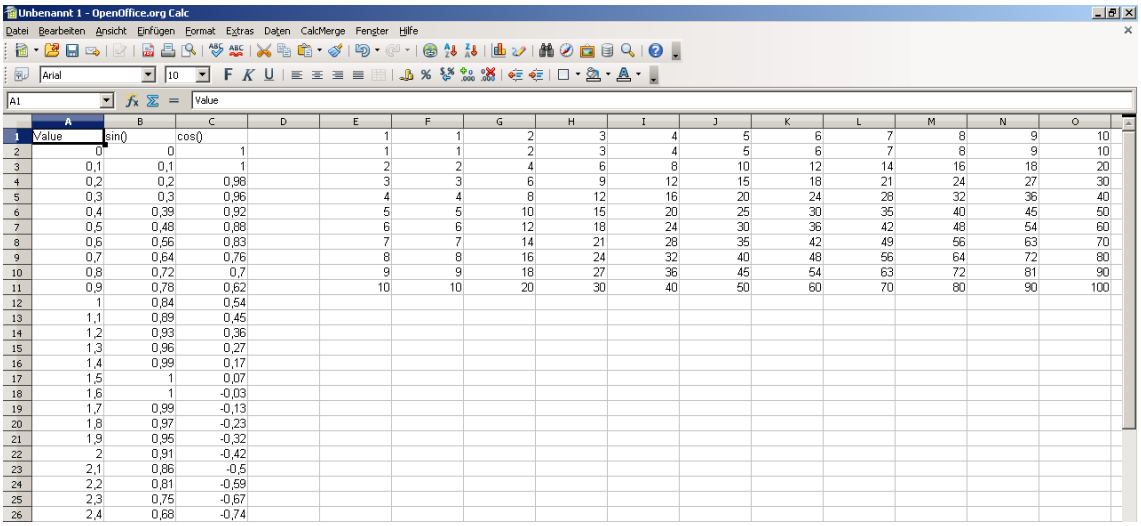
Multiple operations supported by OpenOffice.org Calc combine series of formulas, variables and values and return the result of the functions operation on each value. Calc allows both functions with one variable resulting in a one-dimensional array and functions with two variables which return a two-dimensional matrix. Pitonyak gives an example of how these capabilities can be used in [Pito04, p. 348–350], the Developer's Guide also introduces them [Sun07, p. 663–664]. Listing 4.9 shows an ooRexx variant of a one- and a two-variables multiple operation, which is an application of trigonometric functions and a simple multiplication. The example should be invoked on an empty Calc sheet and results in figure 4.1.

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
  xcc = xsc~getComponentContext()
6 xMCF = xcc~getServiceManager()
  xMSF = xdoc~XMultiServiceFactory
8
  /*select procedure as in Frysak2008*/
10 /*currently selected sheet*/
```

```
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

/**ONE VARIABLE**/
/*first create some input data*/
/*see listing 23 "AutomaticFills.rex" for details*/
oRange = oSheet~XSheetCellRange~getCellRangeByName("A2:A100")
call uno.setCell oSheet,"A1","Value"
call uno.setCell oSheet,"A2",0
oRange~XCellSeries~fillSeries(-
  bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_BOTTOM"),-
  bsf.getStaticValue("com.sun.star.sheet.FillMode","LINEAR"),-
  bsf.getStaticValue("com.sun.star.sheet.FillDateMode","FILL_DATE_DAY"),-
  0.1,6.4)

/*set header for formula*/
call uno.setCell oSheet,"B1","sin()"
call uno.setCell oSheet,"C1","cos()"

/*set formulas*/
call uno.setCell oSheet,"B2","=sin(A2)"
call uno.setCell oSheet,"C2","=cos(A2)"

/*get entire block to operate on*/
oRange = oSheet~XSheetCellRange~getCellRangeByName("A3:C66")

/*get the formula addresses*/
oBlockAddress = oSheet~XSheetCellRange~getCellRangeByName("B2:C2")~XCellRangeAddressable~
    getRangeAddress()

oCellAddress = oSheet~getCellByPosition(0,1)~XCellAddressable~getCellAddress()

oRange~XMultipleOperation~setTableOperation(-
  oBlockAddress,-
  bsf.getStaticValue("com.sun.star.sheet.TableOperationMode","COLUMN"),-
  oCellAddress,oCellAddress)

/**TWO VARIABLES**/
/*first create some input data*/
/*see listing 23 "AutomaticFills.rex" for details*/
call uno.setCell oSheet,"E1","=F1*E2"
call uno.setCell oSheet,"E2","1"
call uno.setCell oSheet,"F1","1"
oRange = oSheet~XSheetCellRange~getCellRangeByName("E2:E11")
oRange~XCellSeries~fillAuto(bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_BOTTOM"),1)

oRange = oSheet~XSheetCellRange~getCellRangeByName("F1:O1")
oRange~XCellSeries~fillAuto(bsf.getStaticValue("com.sun.star.sheet.FillDirection","TO_RIGHT"),1)
oCell=oSheet~getCellByPosition(4,0)
.bsf.dialog~messageBox(uno.getProperties(oCell)~makeArray(" ")~toString)

/*get the referenced cells*/
oRowCellAddress = oSheet~getCellByPosition(5,0)~XCellAddressable~getCellAddress()
oColCellAddress = oSheet~getCellByPosition(4,1)~XCellAddressable~getCellAddress()

/*get entire block to operate on*/
oRange = oSheet~XSheetCellRange~getCellRangeByName("E1:O11")

oRange~XMultipleOperation~setTableOperation(-
  oRange~XCellRangeAddressable~getRangeAddress(),-
  bsf.getStaticValue("com.sun.star.sheet.TableOperationMode","BOTH"),-
  oColCellAddress,-
  oRowCellAddress)

::requires UNO.cls
```

Listing 4.9: 19_MultipleOperations.rex. Demonstrates the multiple operation functionality.

In both cases, at first the values have to be filled (listing 4.9) using the automatic filling procedures as described in listing 4.7. For the one-variable operation (lines 13–44), we define a range from 0.0 to 6.4 bottomwards (line 23), the `FillMode` is `"LINEAR"`, thus a constant 0.1 increase of the variable. The initial formula is set to the cells B2 and C2 containing the trigonometric functions. We then retrieve the whole address block of values and on which the results will be written to (variable `oRange`), ignoring the first line where the formulas and the initial have been set to, which addresses are separately saved in `oBlockAddress` (formulas) and `oCellAddress` (initial value).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Value	sin()	cos()		1	1	2	3	4	5	6	7	8	9	10
2	0	0	1		1	1	2	3	4	5	6	7	8	9	10
3	0,1	0,1	1		2	2	4	6	8	10	12	14	16	18	20
4	0,2	0,2	0,98		3	3	6	9	12	15	18	21	24	27	30
5	0,3	0,3	0,96		4	4	8	12	16	20	24	28	32	36	40
6	0,4	0,39	0,92		5	5	10	15	20	25	30	35	40	45	50
7	0,5	0,48	0,88		6	6	12	18	24	30	36	42	48	54	60
8	0,6	0,56	0,83		7	7	14	21	28	35	42	49	56	63	70
9	0,7	0,64	0,76		8	8	16	24	32	40	48	56	64	72	80
10	0,8	0,72	0,7		9	9	18	27	36	45	54	63	72	81	90
11	0,9	0,78	0,62		10	10	20	30	40	50	60	70	80	90	100
12	1	0,84	0,54												
13	1,1	0,89	0,45												
14	1,2	0,93	0,36												
15	1,3	0,96	0,27												
16	1,4	0,99	0,17												
17	1,5	1	0,07												
18	1,6	1	-0,03												
19	1,7	0,99	-0,13												
20	1,8	0,97	-0,23												
21	1,9	0,95	-0,32												
22	2	0,91	-0,42												
23	2,1	0,86	-0,5												
24	2,2	0,81	-0,59												
25	2,3	0,75	-0,67												
26	2,4	0,68	-0,74												

Figure 4.1.: The multiple operations document after executing listing 4.9. On the left side, the one-variable variant demonstrates calculation of $\sin(x)$ and $\cos(x)$, the right shows a times table.

To sum up, the first line is completely set, its variable and functions addresses saved in variables, as well as the block the results should be written to, including the values. Now the operation can be called, which is accessible via the **`XMultipleOperation`** interface supported by a cell range (line 44). Its only method, `setTableOperation()` expects four parameters and has to be called on the results/values block. The first is a cell range address and links to the formula range, the second gives the operation mode (`com.sun.star.sheet.TableOperationMode`, [Openb, sheet/TableOperationMode]), it codes whether the direction is `"COLUMN"`, `"ROW"` or, in case of two variables, `"BOTH"`. The third and the fourth parameters contain the references to the first and second initial values, the first coding the row and the second the column value. In the one-variable modes, only one of these parameters is evaluated [Openb, sheet/XMultipleOperation].

Similarly, we set up a series of row and column values for the two-variable operation (lines 46–70). The automatic filling creates a column containing whole numbers from 1–10 (line 53)

and a row with whole numbers from 1–7 (line 55). The formula cell is set in the left upper edge of the matrix, tests show that the multiple operations don't work if it is positioned somewhere else. Then the row and column cell addresses are retrieved and saved in `oRowCellAddress` and `oColumnCellAddress`, and the operating block. In this case, it also includes the formula and the initial values (line 64) [Sun07]. In the following table operation execution, the mode is set to `"BOTH"`, and the initial row and column addresses are referenced (line 70).

4.3. Cell Grouping and Connection Visualization

OpenOffice.org Calc offers some capabilities supporting the user in keeping track of their cells as well as focusing on the important document parts, namely the outline and detective features.

4.3.1. Example 20 – Usage of Outlines

The outline or grouping functionality in OpenOffice.org enables the user to combine certain cell ranges, which can either be columns or rows, and hide and show them via easy accessible expansion buttons displayed around the sheet. Both the creation as well as the expansion change can be scripted, as Pitonyak demonstrates [Pito07, chapter 6.15]. Although an execution on an empty Calc sheet is possible, it is recommended to insert some data before, e.g. using listing A.3 of appendix A.1.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

/*first we remove all existing outline groups*/
oSheet~XSheetOutline~clearOutline()

/*params: left,top,right,bottom*/
oRangeRows = oSheet~XSheetCellRange~XCellRange~getCellRangeByPosition(0,0,2,2)
oSheet~XSheetOutline~group(oRangeRows~XCellRangeAddressable~getRangeAddress(),bsf.getStaticValue(
    "com.sun.star.table.TableOrientation","ROWS"))

/*params: left,top,right,bottom*/
oRangeCols = oSheet~XSheetCellRange~XCellRange~getCellRangeByPosition(3,3,5,5)
oSheet~XSheetOutline~group(oRangeCols~XCellRangeAddressable~getRangeAddress(),bsf.getStaticValue(
    "com.sun.star.table.TableOrientation","COLUMNS"))

/*we then hide the rows outline*/
oSheet~XSheetOutline~hideDetail(oRangeRows~XCellRangeAddressable~getRangeAddress())

```

```
::requires UNO.cls
```

Listing 4.10: 20_Outlines.rex. Outlines and their expansion options are set.

The outline functionality is included in the interface **XSheetOutline** of a spreadsheet. After retrieving the active sheet in listing 4.10, we first clear all existing outlines, if any present, by calling `clearOutline()`. Then we create both a group containing rows and one for columns with the help of `group()`, which expects two parameters indicating the affected cell range as the structure `com.sun.star.table.CellRangeAddress` and the orientation, which can be retrieved from `com.sun.star.table.TableOrientation`. It just contains `"ROWS"` and `"COLUMNS"` [Openb, table/TableOrientation]. The `getCellRangeByPosition()` procedure is used to set up the cell range address. Rows and columns of the rectangle range can here be inserted numerically. Of course, other methods of retrieving a cell range are possible, e.g. out of a selection. At any case, a cell range can be converted by the `getRangeAddress()` method of the **XCellRangeAddressable** interface into the required range address structure [Openb, sheet/XCellRangeAddressable].

After the set up, by default the outlines created are expanded, they can be collapsed with the `hideDetail()` method and shown again with `showDetail()`. Additionally, `autoOutline()` tries to automatically find groups depending on the formula references in the cells [Sun07, p. 718].

4.3.2. Example 21 – Detective

Sometimes the user intends to visualize the cellular dependencies within a Calc document. OpenOffice.org therefore provides the detective service, which is able to draw arrows to dependents as well as precedents of a cell. Especially if the user wants to display the whole chain of cells referencing each other, the GUI tool is uncomfortable, as it operates on a stepwise basis adding and deleting one level only. The following script 4.11 creates the arrows to all precedents and afterwards to all descendents of a selected cell with the help of the detective service at once. A preparing script which also selects a suitable cell can be found in appendix A.1, listing A.5.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*get active sheet*/
oController = xdoc~XModel~getCurrentController()
oSheet=oController~XSpreadsheetView~getActiveSheet()

```

```
enumFormula = bsf.getConstant("com.sun.star.table.CellContentType","FORMULA")

/*show precedents of a formula cell - get currently selected cell*/

oSelection = oController~XSelectionSupplier~getSelection()

if \ uno.supportsService(oSelection,SheetCellRange) then
do
  .bsf.dialog~messageBox("Selection must be a cell.")
  exit
end

oCell = oSelection~XCellRange~getCellByPosition(0,0)
stCellAddress = oCell~XCellAddressable~getCellAddress()

/*clear existing arrows*/
oSheet~XSheetAuditing~clearArrows()

/*is it a formula cell?*/
if UNO.areSame(oCell~getType(),enumFormula) then
do
  /*show precedents of a formula cell*/
  loop until ret==0
  ret = oSheet~XSheetAuditing~showPrecedents(stCellAddress)
  end
end
else
do
  .bsf.dialog~messageBox("You must specify a formula cell to show precedents")
end

.bsf.dialog~messageBox("Precedents shown. Now going to show dependents.")
oSheet~XSheetAuditing~clearArrows()

/*show dependents of a formula cell*/
loop until ret==0
ret = oSheet~XSheetAuditing~showDependents(stCellAddress)
end

::requires UNO.cls
```

Listing 4.11: 21_Detective.rex. The script demonstrates the use of the detective.

In listing 4.11 first the selected cells are retrieved. The `getSelection()` method returns an **`XInterface`**, which is dependent on the type of object selected. For cells, the `SheetCellRange` service is supported, so the script checks for it in line 19. Although it is now ensured that only cells are selected, the selection still could be a range compassing various columns and rows. The script therefore selects the cell in the first row and first column of the provided range via `getCellByPosition()`, which expects the row and column number of the cell to select relative to the range [Openb, table/XCellRange], starting with 0. The `getCellAddress()` function is used to retrieve the cell's `com.sun.star.table.CellRangeAddress`, which the detective functions need as input. They are available in the **`XSheetAuditing`** interface implemented by the sheet, and at the beginning all existing arrows are deleted (line 29). Of course, only a formula cell can have precedents, we ensure that via the `getType()` function of the cell and then start the visualization step. `showPrecedents()` draws arrows to these cells directly referenced by the cell given as parameter, but when called repeatedly with the same cell, an additional level of precedent cells is added, thus the pre-precedent in the second call. The boolean return

indicates whether an additional cell reference has been visualized.

The script calls the function within an iteration until all precedents have been highlighted. Then a message box pops up, after confirming, the arrows are again deleted and the same is carried out again for the dependent cells using the `showDependents()` method.

The **`XSheetAuditing`** interface also allows to clear only one level (`hideDependents()`) and to show the cells causing an error with `showErrors()` [Openb, sheet/XSheetAuditing]. At least until OpenOffice.org 3.0, the detective feature is only available for references within the sheet itself [Open05].

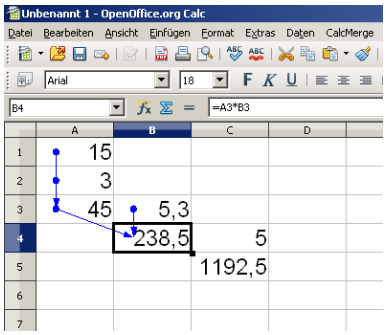


Figure 4.2.: The precedents of the highlighted cells are shown after executing listing 4.11.

5. Selected Generic Pitonyak OpenOffice.org Examples

As OpenOffice.org is built upon re-usable components, some concepts all available for all modules with small modifications only.

5.1. Graphics capabilities

Graphics operations are similar for all OpenOffice.org modules. The section thus demonstrates some graphics scripting and explains modifications necessary to use it in all modules.

5.1.1. Example 22 – Convert Linked Graphics

Graphics in text documents can either be embedded within the document, thus saved in the OpenOffice.org Writer file, or just linked, where the Writer document only contains a URL pointing at the graphics in the file system. The type is decided upon during graphics import. On the one hand, this can be very efficient – the graphics aren't duplicated, and changes immediately affect the document. On the other hand, linked graphics might turn out to be a nightmare, especially when sharing files. Wrong paths or even a not transmitted graphics file cause the graphic to be missing in the document.

When inserting a graphics via the API, by default OpenOffice.org only links the graphic to its source [Pito07, Sun07]. Resultingly, all ooRexx examples available by May 2009, which deal with nearly all program components of OpenOffice.org, only insert graphic links, but not embedded graphics (compare [Gmei08, Frys08, Hinz06, Gund07]).

Apart from manually modifying all graphics, the transformation of a document's graphics from linked to embedded can be done by script, too. Pitonyak presents one capable of processing `GraphicObjectShape` and `TextGraphicObject` graphics in his supplementary Writer document [Pito07, listing 5.29]. Listing 5.1 is an ooRexx transcript of his script.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*counter variables to document changes*/
iEmbedded =0
iLinked=0
iConverted=0

/*two forms of graphic objects are supported*/
s1="com.sun.star.drawing.GraphicObjectShape"
s2="com.sun.star.text.TextGraphicObject"

/*get the document's draw page*/
oDrawPage = xdoc~XDrawPageSupplier~getDrawPage()

/*create the linked array*/
aLinked = .array~new

do i=1 to oDrawPage~XIndexAccess~getCount()
  oGraph = oDrawPage~XIndexAccess~getByIndex(i-1)

  /*look whether object supports GraphicObjectShape service*/
  if (oGraph~XServiceInfo~supportsService(s1) | oGraph~XServiceInfo~supportsService(s2)) then
  do
    if oGraph~XPropertySet~getPropertyValue("GraphicURL")~pos("vnd.sun") <> 0 then
    do
      iEmbedded = iEmbedded+1
    end
    else
    do
      iLinked = iLinked+1
      aLinked~append(oGraph)
    end
  end
end

do i=1 to aLinked~items
  oGraph = aLinked~at(i)
  sName=.DateTime~new~~fullDate~string
  oBitmaps = xMSF~createInstance("com.sun.star.drawing.BitmapTable")
  do while oBitmaps~XNameContainer~XElementAccess~hasByName(sName)
    sName = .bsf.dialog~inputBox("Name" sName "already exists.Please enter alternative link
        display name.",sName)
  end
  oBitmaps~XNameContainer~insertByName(sName,oGraph~XPropertySet~getPropertyValue("GraphicURL"))
  sNewURL = oBitmaps~XNameContainer~XElementAccess~getByName(sName)
  oGraph~XPropertySet~setPropertyValue("GraphicURL",sNewURL)
  iConverted = iConverted+1
end

.bsf.dialog~messageBox("Converted" iConverted "graphics")

::requires UNO.cls
```

Listing 5.1: 22_ConvertLinkedGraphics.rex. Converts all linked to embedded graphics.

After the default connection statements, we first initialize three counter variables, which just document changes processed (lines 9–12). Then two strings are defined, they code the possible graphics types to modify. As examplified in Pitonyak [Pito07, listing 5.29], the script accepts both graphic insertion variants, `TextGraphicObject` and `GraphicObjectShape`, as input (see

section 2.2.4). For easy referencing, two string variables `s1` and `s2` are holding these paths as values. In contrary to Frysak, who used `getGraphicObjects()` [Frys08], we use the Writer draw page to obtain the graphic elements, as it does not only contain all shape objects, but also the base frame objects of the document [Sun07, p. 605]. The draw page is available via `getDrawPage()` method provided by the **`XDrawPageSupplier`** interface of the document's model (line 19) and implements the **`XIndexAccess`** interface, which allows for both obtaining the number of elements (`getCount()`) and accessing a certain element (`getByIndex()`). We iterate over all elements and check whether one of the two graphic formats is supported (line 28). This is necessary as the draw page might also include other, non-graphic shape objects.

Then we look for the existence of the phrase `"vnd.sun"` in the `"GraphicURL"` property of each element to distinguish between already embedded and linked-only images. If an image is embedded, the property starts with `"vnd.sun.star.GraphicObject:"` [Pito07], otherwise it simply includes the source link and the element is appended to the `aLinked` array containing all linked graphics.

The actual conversion is processed by inserting the graphic into a `BitmapTable` service. This table is a collection of internal URLs to graphics embedded into the document. After a URL is added, the graphic is saved internally and the new internal URL can be retrieved via a unique name, as the bitmap table supports the **`XElementAccess`** interface. In this context, this name is only for internal use. We therefore create it automatically by a timestamp. If a computer ever happens to convert two graphics at the same time in milliseconds, and error would occur. It is therefore checked whether the created timestamp is already present and, if so, the user is asked for an alternative label. Then the graphic is added via the `insertByName()` method, after the URL retrieval we reset the `"GraphicURL"` property to the new URL (compare [Foru04]).

For `GraphicObjectShape` types only, an alternative embedding procedure is available [Pito07, listing 5.29]. Here a new object is created, and the most important properties, the graphic URL, size and position, are copied to the new one, which is then inserted at the same document position. The old object is removed with help of `removeTextContent()` (see appendix A, listing A.9). In principle, listing A.9 might replace lines 44–51 of listing 5.1, if only `GraphicObjectShapes` are present (e.g. in a Calc document).

In contrast to Pitonyak, who manages to immediately transform the graphics according to this procedure, ooRexx examples show that problems with the **`XIndexAccess`**-provided element access occur when using the alternative version for `GraphicObjectShape` in listing A.9. Obviously, the numbering in the array changes after an element has been deleted and recreated, therefore we use an array with the elements to change in order to keep the correct references and then iterate over it.

Of course, this script can also be used in other OpenOffice.org applications, as all supply at least one draw page, on which the script can operate. Unfortunately, the retrieval methods of the draw page differ. In Writer with only one draw page, line 19 of listing 5.1 can be used, for Draw and Impress as well as for Calc, listing 5.2 provides different retrieval methods, each line might replace line 19 of listing 5.1. Listing 5.5 examplifies how the script could determine which office module is in usage automatically.

```
/*get the document's draw page*/
/*Calc, Impress, Draw - copy the following line*/
oDrawPage = xdoc~XDrawPagesSupplier~getDrawPages()~XDrawPages~getByIndex(0)
/*Writer - copy the following line*/
oDrawPage = xdoc~XDrawPageSupplier~getDrawPage()
/*Draw and Impress also support*/
oDrawPage = xdoc~XModel~getCurrentController()~XDrawView~getCurrentPage()
```

Listing 5.2: Alternative draw page retrieval methods.

5.1.2. Example 23 – Graphic Size

Inserting a graphic via a script saves a lot of time. Nonetheless, the graphic size has to be set manually in a lot of cases, especially if the graphics are not prepared well regarding resolution or native size. Additionally, the `GraphicObjectShape` type requires a size to be set, as it is derived from the shape classes, which also include vector drawings and the like. Of course, the user normally expects the imported graphics to be scaled proportionally, thus the original size must be known in advance. Listing 5.3 gives an example on how to insert graphics and obtain the correct proportions. It is derived from Pitonyak [Pito07, listings 5.77, 5.78, 5.79; originally developed by Vance Lankhaar]. It works for all OpenOffice.org modules supporting shapes, thus Writer, Impress, Draw and Calc as well as for the formula pages of Base. If the user doesn't feel save with manually specifying paths to filenames, in listing A.10 in appendix A.2 a graphic file selection dialog is provided.

```
/******************/
/**USER MODIFIABLE**/
/******************/
sGraphicURL = "c:\temp\png.png"
/************************/
/**END USER MODIFIABLE**/
/************************/

/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory
s = "com.sun.star.drawing.GraphicObjectShape"

sGraphicURL=uno.convertToURL(sGraphicURL)
oDrawPage = xdoc~XDrawPageSupplier~getDrawPage()
```

```
20
   oGraphic = xMSF~createInstance(s)
22 oGraphic~XPropertySet~setPropertyValue("GraphicURL",sGraphicURL)
   oShape = oGraphic~XShape
24
   oGraphicProvider = xMCF~createInstanceWithContext("com.sun.star.graphic.GraphicProvider",xcc)~
       XGraphicProvider
26 aMediaProperties = uno.CreateArray(.UNO~PROPERTYVALUE,1)
   aMediaProperties[1] = uno.createProperty("URL",sGraphicURL)
28
   /*obtain the graphic's property set*/
30 oGraphicPropertySet = oGraphicProvider~queryGraphicDescriptor(aMediaProperties)~XPropertySet

32 /*returns the "print size" set for a graphic in 100th mm*/
   oGraphicSize = oGraphicPropertySet~getPropertyValue("Size100thMM")
34 iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height")
   iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width")
36
   /*are the sizes set?*/
38 /*if not, we set them at least in the right aspect ratio*/
   if (iGraphicHeight==0 | iGraphicWidth==0) then
40 do
     oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
42   iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height") *25
     iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width") *25
44
     /*ok, the preload mechanism failed - lets retrieve the graphic and then determine its size*/
46   if (iGraphicHeight==0 | iGraphicWidth==0) then
     do
48     oGraphic = oGraphicProvider~queryGraphic(aMediaProperties)
       if oGraphic <> .nil then
50     do
       oGraphicPropertySet = oGraphic~XPropertySet
52     oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
       iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height") *25
54     iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width") *25
       end
56     /*still nothing? -- well, giving up.*/
       else
58     do
         .bsf.dialog~messageBox("Couldn't determine size. Exiting...")
60       exit
       end
62   end
   end
64
   /*set the new size and add the graphic to the draw page*/
66 size=.bsf~new("com.sun.star.awt.Size")
   size~bsf.setFieldValue("height",iGraphicHeight)
68 size~bsf.setFieldValue("width",iGraphicWidth)

70 /*setSize sets the graphic size in 100th mm*/
   oShape~setSize(size)
72 oDrawPage~XShapes~add(oShape~XShape)

74 ::requires UNO.cls
```

Listing 5.3: 23_GraphicSize.rex. Inserts a shape graphic and retains its proportion.

At the beginning, the service needed and the graphics URL (`sGraphicURL`) are set. The drawpage of the Writer document or Base formular is retrieved and a service instantiated. For use with other OpenOffice.org modules where more than one drawpage is existent, one would have to modify this line only, as listing 5.4 shows. Listing 5.5 examplifies how the script could determine which office module is in usage automatically.

```
  oDrawPage = xdoc~XDrawPagesSupplier~getDrawPages()~XDrawPages~getByIndex(0)
2 /*Draw and Impress also support*/
  oDrawPage = xdoc~XModel~getCurrentController()~XDrawView~getCurrentPage()
```

Listing 5.4: Alternative line for listing 5.3 supporting multiple draw pages (Draw, Calc, Impress).

After the graphic URL is set, we instantiate a `com.sun.star.graphic.GraphicProvider` service, which **`XGraphicProvider`** interface is responsible for handling graphic content. The **`queryGraphicDescriptor()`** method expects at least the graphic URL property as parameter and should return an **`XPropertySet`** characterizing the query graphic. This property set includes `"MimeType"`, which contains the mime type of the graphic as well as the graphic's size for pixel graphics [Openb, graphic/XGraphicProvider]. Here two properties are available, `"Size100thMM"` and `"SizePixel"`. The first returns a `com.sun.star.awt.Size` construct containing the defined printing size of the graphics in 100th mm, the last the size in pixels.

Of course, the printing sizes are not set for all graphics. Therefore, after we fail to obtain it expressed as the `"Height"` and `"Width"` properties being 0, we take a look at the pixel size (lines 37–43). Pixel size has nothing to do with printing size, so in this case we can only try to retain the proportions of the images, but not their actual size in mm. By default, the pixel size is multiplied by 25. This is caused by the final **`setSize()`** method of the graphic expecting a value in 100th mm, which typically is not equal to one pixel, as this would lead to a resolution of 2540 dpi. Resultingly, we set a resolution of around 100 dpi.

Interestingly, the retrieval of the pixel size fails in some cases. While it nearly always succeeds for PNG- and GIF-graphics, JPG-images return 0, even when created with OpenOffice.org. This behaviour contradicts API documentation, according to it the **`queryGraphicDescriptor`** procedure is used to query "in the case of pixel graphics, the resulting size after loading" [Openb, graphic/XGraphicProvider].

A workaround is provided by **`queryGraphic()`**, which returns an **`XGraphic`** interface actually holding the graphic. This interface represents the graphic in the `GraphicObjectShape`, and apart from many other properties, its **`XPropertySet`** provides the size in pixels, too. We retrieve the graphic and the interface to it (line 48), and then get the needed properties from the `oGraphic` variable. This version also works for all tested JPG-graphics, but it has to be noted that it is not a prefetch method, the graphic actually gets loaded.

If we still don't have a valid size, the script exits. Otherwise, a `Size` structure is created and filled with the obtained values, later the print size of the graphic is set via **`setSize()`**. In the last step, we add the graphic to the draw page via the **`XShapes`** interface providing the **`add()`** method.

5.1.3. Example 24 – Exports Graphics at Defined Size

The export of all graphics in a document has already been shown by Frysak. He asks for an export directory with the `OfficeFolderPicker` dialog, then accesses all `TextGraphicObject` elements via the `getGraphicObjects()` method and uses the **`XGraphic`** interface of the images to write it out into one file each [Frys08].

We follow a different approach, which is primarily applicable to Impress and Draw. Frysak uses the `storeGraphic()` method of the **`XGraphicProvider`** interface to export images, while we implement a script based on the `GraphicExportFilter` service. Its export capabilities are not restricted to images accessible via the **`XGraphic`** interface, via the service it is also possible to export a drawing page on the whole, a shape or group of shapes [Openb, drawing/GraphicExportFilter]. Although the `TextGraphicObject` images are also part of the drawing page [Sun07, p. 605], the script fails when used to export that type of graphics from Writer. Resultingly, the Writer applicability is restricted to export shape items. The script is derived from an OpenOffice.org codesnippet [SaJS04]. If the user doesn't feel save with manually specifying paths to filenames, in listing A.11 in section A.2 a graphic file selection dialog is provided.

```
/******************/
/**USER MODIFIABLE**/
/******************/
sFileURL = "c:\temp\test_export.png"
/************************/
/**END USER MODIFIABLE**/
/************************/

/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

aFilterData = uno.createArray(.UNO~PROPERTYVALUE,5)
aFilterData[1]=uno.createProperty("PixelWidth",box("I",1000))
aFilterData[2]=uno.createProperty("PixelHeight",box("I",1000))
aFilterData[3]=uno.createProperty("LogicalWidth",box("I",1000))
aFilterData[4]=uno.createProperty("LogicalHeight",box("I",1000))
aFilterData[5]=uno.createProperty("Quality",box("I",90))

sFileUrl = uno.convertToUrl(sFileURL)
xView = xdoc~XModel~getCurrentController()
xSelection = xView~XSelectionSupplier~getSelection()

/*if we have a text document, we need to reduce the contained elements to XShapes*/
if xdoc~uno.supportsService("TextDocument") | xdoc~uno.supportsService("SpreadsheetDocument")
    then
do
  /*only shapes selected?*/
  if \ uno.getInterfaceNamesViaReflection(xSelection)~makeArray(" ")~hasItem("
      com.sun.star.drawing.XShapes") then
  do
    .bsf.dialog~messageBox("No shapes selected. Exiting.")
    exit
  end
end
```

```
38 else
   do
40   /*xSelection contains the selected objects - if nothing is selected, the interface name is void
         */
     if uno.getInterfaceName(xSelection)=="void" then
42   do
       xSelection = xView~XPropertySet~getPropertyValue("CurrentPage")
44   end
   end
46
   xExporter = xMCF~createInstanceWithContext("com.sun.star.drawing.GraphicExportFilter",xcc)
48 xExporter~XExporter~setSourceDocument(xSelection~XComponent)

50 aArgs=uno.createArray(.UNO~PROPERTYVALUE,3)
   aArgs[1]=uno.createProperty("MediaType","image/jpeg")
52 aArgs[2]=uno.createProperty("URL",sFileURL)
   aArgs[3]=uno.createProperty("FilterData",aFilterData)
54
   a =xExporter~XFilter~filter(aArgs)
56 .bsf.dialog~messageBox(a)

58 ::requires UNO.cls
```

Listing 5.5: 24_ExportGraphicsSize.rex. A script exporting one selected document part.

At first, we define the filter data for our export (lines 17–22). The options available are dependent on the media type, in our case it is `"image/jpg"`, which is later set before exporting the document (line 51). We choose to set `"PixelHeight"` and `"PixelWidth"` which determine the pixel size of the exported image as well as `"LogicalHeight"` and `"LogicalWidth"`, which express the height and width of the image in 100^{th} mm. Resultingly, we should have a print size of 10 mm, nonetheless, an import into GIMP (GNU Image Manipulation Program) as well as an reimport using listing 5.3 shows that this feature doesn't seem to work correctly. Another option is the `"Quality"`, which allows to influence the JPG compression and the image size (compare [SaJS04]).

Then we retrieve the current selection to export via the **`XSelectionSupplier`** interface of the model controller and the **`getSelection`**`()` method. Before exporting it, we have to ensure that only a draw page, a shape or a group of shapes are selected. In the case of Draw and Impress, any selection can be exported as it is on a draw page. Problems may only occur in Calc, where cells can be selected, or in Writer, where e.g. normal text or tables can be chosen. Therefore, we first check whether the script was executed for a Calc or Writer document by looking for their typical services, `"TextDocument"` and `"SpreadsheetDocument"` (line 29). A Base formular is also affected, it also implements the `"TextDocument"` service. Lines 29–37 then handle this special case. The returned selection is of type **`XInterface`** and points to one object, depending on the current selection [Sun07, p. 627]. This object implements various interfaces, which are typical for the type of object selected. As at the same time only one type of objects can be selected (e.g. table cells or text), we can derive whether the current selection consists of shapes by looking for the `"XShapes"` interface (line 32). Alternatively, a check for supported service would also be possible (compare listing 3.1–3.3). If not, the script exits; this is also the case

if nothing is selected, as it is equal to a collapsed text selection and therefore implements the `XIndexAccess` interface [Sun07, p. 627].

For Draw and Impress, we check if anything is selected via the interface name of the selection. It is `"void"` in case of an empty selection (line 41). We then assume the user intends to export the whole page, which is chosen via the `XPropertySet` of the document controller and the `"CurrentPage"` property.

In the last step, the graphic export filter is instantiated. The source document needs to be set, it has to be of type `XComponent`, which is supported by draw pages and shapes (line 48). Its actual filter method provided by the interface `XFilter`, `filter()`, requires a media descriptor as parameter. The export filter supports only three parts of that powerful service, `"URL"` which sets the destination, `"MediaType"` to modify the type of the output and `"FilterData"` used to apply additonal parameters depending on the media type. Then we can trigger the export by issuing the `filter()` method (line 55), and a message box states 1 if the process was successful.

5.1.4. Example 25 – Combine Graphics and Export – Draw and Impress

OpenOffice.org Draw and Impress allow some more complex graphic operations, an example is shown in listings 5.6–5.8, they only work in OpenOffice.org Draw and OpenOffice.org Impress. Here, a number of input graphics is specified, which get combined to one single graphic and again exported. Apart from the input files, the user must also tell the script the raster size, e.g. by defining a 2×2 matrix which holds exactly 4 images and the output file to write the newly created graphic to. The script is based on the graphic operations already demonstrated in listings 5.3 and 5.5.

If the user doesn't want to specify file names manually, a GUI is provided in appendix A, listing A.12, supporting the user with the inputs.

```
/******************/
/**USER MODIFIABLE**/
/******************/

/**A GUI IS ALSO PROVIDED, SEE TEXT**/

/*specify output filename including .jpg extension*/
filename = "c:\temp\export.jpg"

/*specify an array of input image files*/
imports = .array~new()
imports~append("c:\temp\png.png")
imports~append("c:\temp\test.png")

/*specify raster columns*/
raster_x=2

/*specify raster rows*/
raster_y=1
```

```
20
   /**********************/
22 /**END USER MODIFIABLE**/
   /**********************/
24
   /*standard header*/
26 xsc = uno.getScriptContext()
   xdt = xsc~getDesktop()
28 xdoc = xsc~getDocument()
   xcc = xsc~getComponentContext()
30 xMCF = xcc~getServiceManager()

32 /*conversions and necessary checks*/
   filename = uno.convertToUrl(filename)
34 do i=1 to imports~items
   imports~put(uno.convertToUrl(imports~at(i)),i)
36 end

38 if \ (imports~items == raster_x*raster_y) then
   .bsf.dialog~messageBox("Error. Image number and raster sizes don't match")
40
   ret = executeCombine(imports,filename,raster_y,raster_x,xdoc,xMCF,xcc)
42
   ::requires UNO.cls
```

Listing 5.6: 25_CombineAndExport.rex, part one. The script shows the parameter definitions required.

The script part in listing 5.6 is primarily necessary for defining the various parameters. At first, an output file name must be specified (line 8). The path names to the input files are then put into one array (lines 11–13), if additional images should be taken into account, the user just has to copy line 13 and define a new file name. Line 16 and 19 contain the variables setting the matrix dimensions, `raster_x` specifies the columns of the resulting images, `raster_y` the rows.

To get OpenOffice.org-compatible path names, the `uno.convertToUrl()` method is applied to all file names specified, which returns a platform-dependent qualified file name as URL [Flat08b], additionally a check is performed whether the matrix size equals the number of input images. Finally, the `executeCombine()` method is called, which is listed in listings 5.7–5.8 to perform the actual conversion.

```
   ::routine executeCombine
2  use arg imports exportfile raster_y raster_x xdoc xMCF xcc

4  xMSF = xdoc~XMultiServiceFactory

6  dpi = 300

8  dp100thmm = dpi/(2.54*10*100)

10 /*create a new drawpage*/
   oDrawPages = xdoc~XDrawPagesSupplier~getDrawPages()
12 iDrawPageCount = oDrawPages~XIndexAccess~getCount()
   oDrawPages~insertNewByIndex(iDrawPageCount)
14 oDrawPage = oDrawPages~XIndexAccess~getByIndex(iDrawPageCount) --XDrawPage

16 /*set pictures at the appropriate positions*/
   oGraphicProvider = xMCF~createInstanceWithContext("com.sun.star.graphic.GraphicProvider",xcc)~
       XGraphicProvider
18 pixelwidths = .array~new
   pixelheights = .array~new
```

```
heightmax = 0
widthmax = 0

do i=1 to imports~items
  sGraphicURL = imports~at(i)
  aMediaProperties = uno.CreateArray(.UNO~PROPERTYVALUE,1)
  aMediaProperties[1] = uno.createProperty("URL",sGraphicURL)
  oGraphicPropertySet = oGraphicProvider~queryGraphicDescriptor(aMediaProperties)~XPropertySet

  /*obtain the pixel sizes*/
  oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
  pixelwidths[i] = oGraphicSize~bsf.getFieldValue("Width")
  pixelheights[i] = oGraphicSize~bsf.getFieldValue("Height")

  /*no size contained in SizePixel property?*/
  if pixelwidths[i]==0 | pixelheights[i]==0 then
  do
  oGraphic = oGraphicProvider~queryGraphic(aMediaProperties)
    if oGraphic <> .nil then
    do
    oGraphicPropertySet = oGraphic~XPropertySet
    oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
    pixelheights[i] = oGraphicSize~bsf.getFieldValue("Height")
    pixelwidths[i] = oGraphicSize~bsf.getFieldValue("Width")
    end
    /*still nothing? -- well, giving up.*/
    else
    do
      .bsf.dialog~messageBox("Couldn't determine graphic object size.")
      return 0
    end
  end

  if pixelwidths[i]>widthmax then
    widthmax = pixelwidths[i]

  if pixelheights[i]>heightmax then
    heightmax = pixelheights[i]

end

/*calculate the raster size*/
rasterwidth = widthmax/dp100thmm
rasterheight = heightmax/dp100thmm

/*we need a background shape*/
oBackground = xMSF~createInstance("com.sun.star.drawing.RectangleShape")
oBackground~XPropertySet~setPropertyValue("LineStyle",bsf.getStaticValue("
    com.sun.star.drawing.LineStyle","NONE"))
oBackground~XPropertySet~setPropertyValue("FillStyle",bsf.getStaticValue("
    com.sun.star.drawing.FillStyle","NONE"))

/*set background size*/
width = rasterwidth*raster_x
height = rasterheight*raster_y
oBackgroundSize = .bsf~new("com.sun.star.awt.Size")
oBackgroundSize~bsf.setFieldValue("Height",trunc(height))
oBackgroundSize~bsf.setFieldValue("Width",trunc(width))
oBackground~XShape~setSize(oBackgroundSize)

/*set background position*/
oBackgroundPos=.bsf~new("com.sun.star.awt.Point")
oBackgroundPos~bsf.setFieldValue("X",0)
oBackgroundPos~bsf.setFieldValue("Y",0)
oBackground~XShape~setPosition(oBackgroundPos)
oDrawPage~XShapes~add(oBackground~XShape)
```

Listing 5.7: 25_CombineAndExport.rex, part two. The script shows the image size detection and the background setting.

Listing 5.7 containing the first part of the combining routing `executeCombine()` first determines the variable value `dp100thmm`, reading "dots per 100th mm". Unfortunately, only a few images specify their print size in mm, resultingly we have to assume a resolution. In this case, it is set to 300. Then a new `DrawPage`, thus a new page in OpenOffice.org Draw, is inserted via the `insertNewByIndex()` method requiring an index, which is the number of the current pages determined via `getCount()` provided by `XIndexAccess`, as the numbering starts with 0. We define an array holding the sizes of the images (`pixelwidths` and `pixelheights`) as well as two variables which are later set to the maximum height and width of all graphics, which is needed to determine the raster size of the matrix. The image size is then retrieved for all graphics similar as in listing 5.3 (lines 23–59) and saved if it exceeds the current maximum.

After the raster size (lines 61–63) is fixed, the size of the whole combined graphic is known, it is `raster_x` times the raster width in width and `raster_y` times the raster height in height. For this size, a background rectangle is inserted. This is needed to provide a correctly scaled output. All images smaller than the largest are inserted in the middle of their "cell", leaving an empty border. Therefore, the export filter would not include these border areas if we had no background rectangle, as there is simply nothing defined on the draw page. Insertion of a rectangle shape and its modifications in OpenOffice.org Draw was demonstrated by Scholz [Scho07], we set both the fill and line style to `"NONE"`, leaving an in principle invisible shape. Afterwards, the rectangle is set to the graphics size and positioned at the very left upper corner of the page (lines 70–82), where also the combined image has its root, and added to the page (line 83).

```
  /*iterate and set positions*/
2 pageoffsetheight = 0
  do j=1 to raster_y
4   pageoffsetwidth = 0
    do k=1 to raster_x
6     i=(j-1)*raster_x+k
      sGraphicURL=imports[i]
8     oGraphic = xMSF~createInstance("com.sun.star.drawing.GraphicObjectShape")
      oGraphic~XPropertySet~setPropertyValue("GraphicURL",sGraphicURL)
10    oShape = oGraphic~XShape

12    height = pixelheights[i]/dp100thmm
      width = pixelwidths[i]/dp100thmm
14
      /*set the new size*/
16    size=.bsf~new("com.sun.star.awt.Size")
      size~bsf.setFieldValue("height",trunc(height))
18    size~bsf.setFieldValue("width",trunc(width))
      oShape~setSize(size)
20
      /*calculate offset*/
22
      customoffsetwidth = pageoffsetwidth + ((rasterwidth - width)/2)
24    customoffsetheight = pageoffsetheight + ((rasterheight - height)/2)

26    /*set the new position*/
      pos=.bsf~new("com.sun.star.awt.Point")
28    pos~bsf.setFieldValue("X",trunc(customoffsetwidth))
      pos~bsf.setFieldValue("Y",trunc(customoffsetheight))
```

```
30      oShape~setPosition(pos)

32      oDrawPage~XShapes~add(oShape)

34      pageoffsetwidth = pageoffsetwidth + rasterwidth
      end
36    pageoffsetheight = pageoffsetheight + rasterheight
    end
38
    /*export to one image file*/
40  xSelectionSupplier = xdoc~XModel~getCurrentController~XSelectionSupplier
    /*select all XShapes of the current page - we want to export all of them*/
42  xSelectionSupplier~select(oDrawPage~XShapes)

44  /*get the selection*/
    xSelection = xSelectionSupplier~getSelection()
46
    /*calculate the size*/
48  size_x = trunc(pageoffsetwidth*dp100thmm)
    size_y = trunc(pageoffsetheight*dp100thmm)
50
    /*setup the filter*/
52  aFilterData = uno.createArray(.UNO~PROPERTYVALUE,5)
    aFilterData[1]=uno.createProperty("PixelWidth",box("I",size_x))
54  aFilterData[2]=uno.createProperty("PixelHeight",box("I",size_y))
    aFilterData[3]=uno.createProperty("LogicalWidth",box("I",trunc(pageoffsetwidth)))
56  aFilterData[4]=uno.createProperty("LogicalHeight",box("I",trunc(pageoffsetheight)))
    aFilterData[5]=uno.createProperty("Quality",box("I",90))
58
    /*get the GraphicExportFilter*/
60  xExporter = xMCF~createInstanceWithContext("com.sun.star.drawing.GraphicExportFilter",xcc)
    xExporter~XExporter~setSourceDocument(xSelection~XComponent)
62
    aArgs=uno.createArray(.UNO~PROPERTYVALUE,3)
64  aArgs[1]=uno.createProperty("MediaType","image/jpeg")
    aArgs[2]=uno.createProperty("URL",exportfile)
66  aArgs[3]=uno.createProperty("FilterData",aFilterData)

68  ret = xExporter~XFilter~filter(aArgs)
    return ret
```

Listing 5.8: 25_CombineAndExport.rex, part three. This part iterates over the images, positions them and finally exports them.

The second part of `executeCombine()` in listing 5.8 iterates over all raster positions determined via `raster_x` and `raster_y`, from the `i` and `j` variables the current linear position number is determined (line 6). A `GraphicObjectShape` object is generated as demonstrated by Burger [Burg06], [Gund07] or Frysak [Frys08], explained for the similarly handled Writer insertion in section 2.2.4. Then the size of the graphic is set as shown in listing 5.3 in 100^{th} mm via the correction factor, and in dependence of the current matrix cell and the raster size, the image is positioned in the middle of the "cell" and added to the draw page (lines 12–32).

After all images are added, the whole draw page is selected via the **`XSelectionSupplier`** interface of the current controller. First, we select by calling its **`select`**`()` method and `oDrawPage` `~`**`XShapes`** holding all shapes of the draw page `oDrawPage` as parameter, then the selection is retrieved using **`getSelection`**`()`. In line 48, the image size is determined via the `pageoffset` variables, which at that time point at the right lower corner of all inserted images. As the left upper corner is at (0,0), the right lower corner codes the image size. Then the filter is set up.

Its properties are dependent on the media type set for the actual export, in our case `"image/jpg"`. Therefore, we set the `"PixelWidth"` and `"PixelHeight"` as well as `"LogicalWidth"` and `"LogicalHeight"` properties and the JPG-specific `"Quality"` property giving the compression rate, they provide the optional `"FilterData"` property of the filter properties, which are a media descriptor (compare [Openb, document/MediaDescriptor]) and specify basic export options. Thus, the export procedure is similar to listing 5.5, and 1 is returned in case of success. Figure 5.1 demonstrates the selected shapes which are exported.

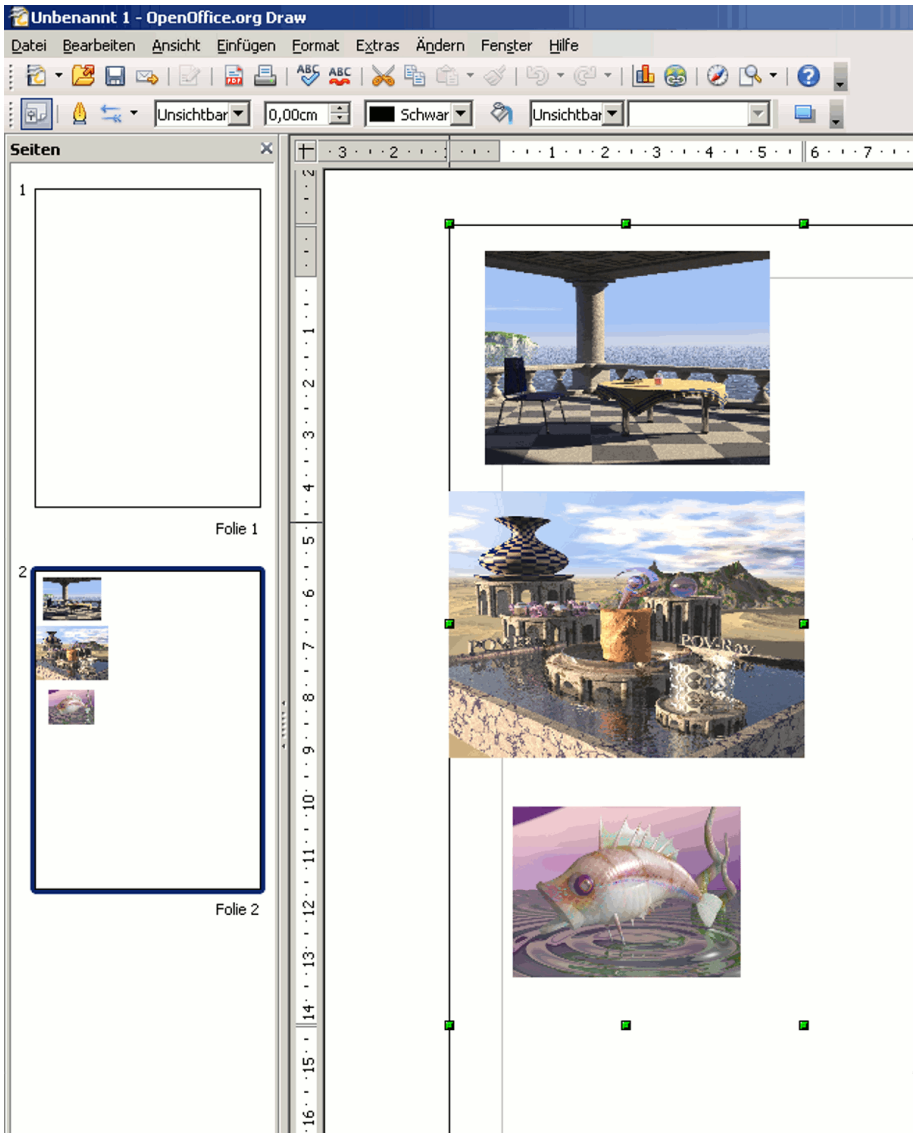


Figure 5.1.: Three graphics as they are exported into one file.

6. Conclusion and Outlook

OpenOffice.org, an open-source alternative to commercial office suite packages, can also be scripted. A splendid work provided by Andrew Pitonyak [Pito04] provides many examples in ooBasic, nonetheless, OpenOffice.org flexible scripting framework also allows the usage of other languages. It is therefore important to supply examples demonstrating a more general view on macro programming, which can also be exploited by users preferring Python, Tcl or ooRexx.

The new examples proved once again the capability of a cooperation between ooRexx and OpenOffice.org, which is possible via the BSF4REXX library. This library also provides special routines which ease the programming of macros a lot.

A special focus was given on text cursors, which are a very basic operation mode needed for nearly all tasks. Additionally, the use of text fields and the integration of Math was demonstrated for Writer. Concerning OpenOffice.org Calc, the work provided some examples on functionalities offered by cell ranges as well as the usage of functions operating on more than one cell. The high compatibility of concepts within the office suite is examplarily demonstrated by exploiting the similarity in graphics operation.

Although many literature examples are available, there are still blank chapters to be addressed. OpenOffice.org provides a multi-platform compatibility – it would be very interesting to further investigate how OpenOffice.org multi-module scripts can be set up, to further dig into commonalities and differences between the module, with the aim of producing not only platform-independent, but also module-independent macros for common tasks.

A. Supplementary Scripts

A.1. Generating Scripts to Fulfil Requirements

A.1.1. Generating Text for Example 02

The following listing inserts some text into an empty Writer document. Principles on text insertion can be found in section 2.2.2.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()

xdoc = xsc~getDocument()

xCursor = xdoc~XTextDocument~getText()~createTextCursor()
xCursor~collapseToStart()
xCursor~XTextRange~setString("Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
     nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad
      minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea
      commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
     molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto
       odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait
     nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy
     nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim
     veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea
     commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
     molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto
     odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla
      facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
      quod mazim placerat facer possim assum.")

::requires UNO.cls
```

Listing A.1: ad2_createText.rex. Creates some text in an empty Writer document.

A.1.2. Generating Database for Example 11

As for a form letter (example 11) a database containing the address data is compulsory, a script creating the necessary files is provided. The following script first asks for a *.odb file to store the OpenOffice.org Base database in, then creates the database and a table “addresses” with the fields as described in example 11. Additionally, it registers the datasource within

OpenOffice.org, making an access easy. Details on implementing Base scripts can be found in Andrew Pitonyak's supplementary macro document dealing exclusively with Base [Pito09] as well as in Schmid's bachelor course paper [Schm07]. The "Save As"-dialog is derived from the example presented in the Developer's Guide [Sun07, p. 1160]. This listing must be executed from within Writer document, so the focus should not be set to the Datasources window or the like!

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

oFilePicker = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")

aListAny = bsf.createArray("Short.class",1)
aListAny[1]=box("short",bsf.getStaticValue("com.sun.star.ui.dialogs.TemplateDescription","
     FILESAVE_AUTOEXTENSION"))
oFilePicker~XInitialization~initialize(aListAny)
xFileManager = oFilePicker~XFilterManager~appendFilter("OpenOffice.org Base","*.odb")

file=.nil
oFilePicker~XExecutableDialog~execute()
file = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)

if file==.nil then
do
 .bsf.dialog~messageBox("No file selected - exiting".)
 exit
end

oSimpleFileAccess = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ucb.SimpleFileAccess")
if oSimpleFileAccess~XSimpleFileAccess2~exists(file) then
do
  signal on ANY
  oSimpleFileAccess~XSimpleFileAccess2~kill(file)
end


table="address"
name="addresses"

oDatabaseContext = xMCF~createInstanceWithContext("com.sun.star.sdb.DatabaseContext",xcc)
xSingleServiceFactory = oDatabaseContext~XSingleServiceFactory
oDataSource=xSingleServiceFactory~createInstance()

oDataSource~XPropertySet~setPropertyValue("URL","sdbc:embedded:hsqldb")

/*get the document*/
oDatabaseDocument = oDataSource~XDocumentDataSource~getDatabaseDocument
oDatabaseDocument~XStorable~storeAsURL(file,uno.createArray(.UNO~PROPERTYVALUE,0))

xSingleServiceFactory~XNamingService~registerObject(name,oDataSource)

oDatabase = oDatabaseContext~XNameAccess~getByName(file)

/*now connect to the DB*/
oConnection = oDatabase~XDataSource~getConnection("","")
oTables = oConnection~XTablesSupplier~getTables()

if oTables~XNameAccess~hasByName(table) then
do
  oTables~XDrop~dropByName(table)
```

```
end

oTableDescriptor = oTables~XDataDescriptorFactory~createDataDescriptor()
oTableDescriptor~XPropertySet~setPropertyValue("Name",table)

oColumns = oTableDescriptor~XColumnsSupplier~getColumns()
oColumnDescriptor = oColumns~XDataDescriptorFactory~createDataDescriptor()

oColumnDescriptor~XPropertySet~setPropertyValue("Name","ID")
oColumnDescriptor~XPropertySet~setPropertyValue("Description","Primary Key")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",10))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.true))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","INTEGER")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)

oColumnDescriptor~XPropertySet~setPropertyValue("Name","First Name")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","VARCHAR")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)


oColumnDescriptor~XPropertySet~setPropertyValue("Name","Last Name")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","VARCHAR")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)


oColumnDescriptor~XPropertySet~setPropertyValue("Name","Street")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","VARCHAR")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)


oColumnDescriptor~XPropertySet~setPropertyValue("Name","ZIP")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","VARCHAR")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)


oColumnDescriptor~XPropertySet~setPropertyValue("Name","City")
oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.ColumnValue","NO_NULLS")))
oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
     com.sun.star.sdbc.DataType","VARCHAR")))
oColumns~XAppend~appendByDescriptor(oColumnDescriptor)


oColumnDescriptor~XPropertySet~setPropertyValue("Name","Sex")
```

```
115 oColumnDescriptor~XPropertySet~setPropertyValue("Precision",box("integer",100))
    oColumnDescriptor~XPropertySet~setPropertyValue("IsNullable",box("integer",bsf.getStaticValue("
        com.sun.star.sdbc.ColumnValue","NO_NULLS")))
117 oColumnDescriptor~XPropertySet~setPropertyValue("IsAutoIncrement",box("boolean",.false))
    oColumnDescriptor~XPropertySet~setPropertyValue("Type",box("integer",bsf.getStaticValue("
        com.sun.star.sdbc.DataType","TINYINT")))
119 oColumns~XAppend~appendByDescriptor(oColumnDescriptor)

121 oTables~XAppend~appendByDescriptor(oTableDescriptor)

123 oDatabaseDocument = oDatabase~XDocumentDataSource~getDatabaseDocument
    oDatabaseDocument~XStorable~store()
125
    sSQL = 'INSERT INTO "address"("First Name","Last Name","Street","ZIP","City","Sex") VALUES
        (?,?,?,?,?,?)'
127 oStatement = oConnection~XConnection~prepareStatement(sSQL)

129 oStatement~XParameters~setString(1,"Christoph")
    oStatement~XParameters~setString(2,"Waglechner")
131 oStatement~XParameters~setString(3,"Augasse 2-6")
    oStatement~XParameters~setString(4,"1090")
133 oStatement~XParameters~setString(5,"Wien")
    oStatement~XParameters~setString(6,0)
135 oStatement~XPreparedStatement~executeUpdate()

137 oStatement~XParameters~setString(1,"Max")
    oStatement~XParameters~setString(2,"Mustermann")
139 oStatement~XParameters~setString(3,"Billrothstraße 1")
    oStatement~XParameters~setString(4,"1051")
141 oStatement~XParameters~setString(5,"Wien")
    oStatement~XParameters~setString(6,0)
143 oStatement~XPreparedStatement~executeUpdate()

145 oStatement~XParameters~setString(1,"Elisabeth")
    oStatement~XParameters~setString(2,"Testerin")
147 oStatement~XParameters~setString(3,"Genzgasse 3/1/2")
    oStatement~XParameters~setString(4,"7000")
149 oStatement~XParameters~setString(5,"Eisenstadt")
    oStatement~XParameters~setString(6,1)
151 oStatement~XPreparedStatement~executeUpdate()

153 oConnection~XCloseable~close()
    oDatabaseDocument~XCloseable~close(.true)
155
    .bsf.dialog~messageBox("File" file "and basic datasets created, registered as" name ".")
157
    exit 0
159
    ANY:
161   .bsf.dialog~messageBox("Could not delete file. Maybe opened? Exiting.")
      exit
163
    ::requires UNO.cls
```

Listing A.2: ad11_createDatabase.rex. Creates the necessary database and some sample data for listing 3.15.

A.1.3. Generating Calc Data for Example 13

The following listing supports the user with inserting some data in an empty Calc sheet, as it is useful for executing some Calc examples. Basics on how to deal with these tasks can be found

in section 2.3.1.

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*get current sheet*/
oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

/*set title row*/
call uno.setCell oSheet,0,0,"Name"
call uno.setCell oSheet,1,0,"Number1"
call uno.setCell oSheet,2,0,"Number2"
call uno.setCell oSheet,3,0,"Number3"

do i=1 to 9
first = D2C(RANDOM(65,90))
second = D2C(RANDOM(97,122))
third = D2C(RANDOM(97,122))
fourth =D2C(RANDOM(97,122))
name = first||second||third||fourth
call uno.setCell oSheet,0,i,name

call uno.setCell oSheet,1,i,RANDOM(0,10)
call uno.setCell oSheet,2,i,RANDOM(0,10)
call uno.setCell oSheet,3,i,RANDOM(0,10)
end

::requires UNO.cls
```

Listing A.3: ad13_createData.rex. Creates the necessary data for some Calc examples.

A.1.4. Generating Styles for Example 14

In order to execute example 14, two cell styles “red” and “green” are needed. The script creates both styles, provided they are not present at execution time. Frysak has already demonstrated how to manage and add styles, the listing is derived from his macro [Frys08].

```
/*Helper script creating the styles necessary for example 14*/
/*Styles creation procedure taken from Frysak, Josef: Automating Open Office - ooRexx Nutshells,
    2008*/
/*Properties taken from Hinz, Michael: OpenOffice.org Calc Automation Using ooRexx, 2006*/

xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

/*search for style name*/
x_StyleFamiliesSupplier = xdoc~XStyleFamiliesSupplier
x_StyleFamilies = x_StyleFamiliesSupplier~getStyleFamilies()
s_StyleFamily = x_StyleFamilies~getByName("CellStyles")
x_NameAccess = s_StyleFamily~XNameAccess

if \ x_NameAccess~hasByName("red") then
do
```

```
21    oCellStyle = xMSF~createInstance("com.sun.star.style.CellStyle")
      x_NameAccess~insertByName("red", oCellStyle)
23    oCellStyle~XPropertySet~setPropertyValue("CellBackColor",box("int","990000"x~c2d))
      oCellStyle~XPropertySet~setPropertyValue("IsCellBackgroundTransparent",box("boolean",.false))
25    oCellStyle~XPropertySet~setPropertyValue("CharColor",box("int","FFFFFF"x~c2d))
    end
27
    if \ x_NameAccess~hasByName("green") then
29  do

31    oCellStyle = xMSF~createInstance("com.sun.star.style.CellStyle")
      x_NameAccess~insertByName("green", oCellStyle)
33    oCellStyle~XPropertySet~setPropertyValue("CellBackColor",box("int","009900"x~c2d))
      oCellStyle~XPropertySet~setPropertyValue("IsCellBackgroundTransparent",box("boolean",.false))
35    oCellStyle~XPropertySet~setPropertyValue("CharColor",box("int","FFFFFF"x~c2d))
    end
37
    ::requires UNO.cls
```

Listing A.4: ad14_createStyles.rex. Creates the necessary styles for listing 4.2.

A.1.5. Generating Formula Cells for Example 21

For example 21, a network of formula references is needed. To ease the preparations for running the script, the listing presented fills an empty Calc sheet as required. Information on setting cell values can be found in section 2.3.1.

```
    /*standard header*/
2   xsc = uno.getScriptContext()
    xdt = xsc~getDesktop()
4   xdoc = xsc~getDocument()
    xcc = xsc~getComponentContext()
6   xMCF = xcc~getServiceManager()
    xMSF = xdoc~XMultiServiceFactory
8
    /*get current sheet*/
10  oSheet = xdoc~XModel~getCurrentController()~XSpreadsheetView~getActiveSheet()

12  call uno.setCell oSheet,0,0,15
    call uno.setCell oSheet,0,1,3
14  call uno.setCell oSheet,0,2,"=A1*A2"

16  call uno.setCell oSheet,1,2,5.3
    call uno.setCell oSheet,1,3,"=A3*B3"
18
    call uno.setCell oSheet,2,3,5
20  call uno.setCell oSheet,2,4,"=B4*C4"

22  oCursor = oSheet~createCursor()
    oCellRange = oCursor~XCellCursor~XCellRange~getCellRangeByPosition(1,3,1,3)
24  --.bsf.dialog~messageBox(uno.getInterfaceNamesViaReflection(oSheet)~makeArray(" ")~toString)
    oCursor = oSheet~XSpreadsheet~createCursorByRange(oCellRange~XSheetCellRange)
26  oSelection=xdoc~XModel~getCurrentController()~XSelectionSupplier~select(oCursor)

28  ::requires UNO.cls
```

Listing A.5: ad21_createFormulaCells.rex. Creates formula cells necessary for example 21.

A.2. Alternative Ways and Supplements

A.2.1. Supplement Literature Example 01 – Graphic Insertion Graphic File Selection

Information on the listing can be found in section 2.2.2. The only addition the listing below provides is a graphic file selection dialog derived from Frysak [Frys08] and the Developer's Guide [Sun07, p. 1160].

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF=xcc~getServiceManager()

call bsf.import "com.sun.star.beans.StringPair","StringPair.class"
filters=bsf.createArray(.StringPair.class,3)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","GIF")
sp~bsf.setFieldValue("Second","*.gif")
filters~put(sp,1)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","JPG")
sp~bsf.setFieldValue("Second","*.jpg;*.jpeg")
filters~put(sp,2)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","PNG")
sp~bsf.setFieldValue("Second","*.png")
filters~put(sp,3)

sGraphicURL = .nil
do while sGraphicURL==.nil
oFilePicker=xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")
oFilePicker~XFilePicker~setMultiSelectionMode(.false)
oFilePicker~XFilterGroupManager~appendFilterGroup("Graphics",filters)
oFilePicker~XFilePicker~XExecutableDialog~execute()
sGraphicURL = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)
end

/* create the TextObject and the TextCursor */
xTextDocument = xdoc~XTextDocument
xText = XTextDocument~getText()
xTextCursor = xText~createTextCursor()
xDMsf = xTextDocument~XMultiServiceFactory

size = .bsf~new("com.sun.star.awt.Size")
size~Height = 1160
size~Width = 4000

/****GraphicObjectShape****/
/*Burger 2006*/
oGraph = xDMsf~createInstance("com.sun.star.drawing.GraphicObjectShape")
xGraph = oGraph~XShape
xGraph~setSize(size)
xPropertySet=xGraph~XPropertySet
xPropertySet~setPropertyValue("GraphicURL", sGraphicURL)
xText~insertTextContent(xText~getEnd,oGraph~XTextContent, .false)

/****TextGraphicObject****/
/*Gmeiner 2008*/
oTextGraphic = xDMsf~createInstance("com.sun.star.text.TextGraphicObject")
xGraphicProperties = oTextGraphic~XPropertySet
```

```
xGraphicProperties~setPropertyValue("GraphicURL", sGraphicURL)
xGraphicProperties~setPropertyValue("AnchorType",bsf.getConstant("
     com.sun.star.text.TextContentAnchorType", "AS_CHARACTER"))
xGraphicProperties~setPropertyValue("Size",size)
xText~insertTextContent(xText~getEnd, oTextGraphic~XTextContent, .false)

::requires UNO.cls
```

Listing A.6: litex01b_GraphicInsertionGUI.rex. Alternative way of retrieving a range address.

A.2.2. Supplement Example 11 – Form Letter Graphic File Selection

The following script provides a graphic file selection dialog setting the document path to which the created form letter should be saved to.

```
xsc = uno.getScriptContext()
if (xsc <> .nil) then
do
-- invoked by OOo as a macro
xcc = xsc~getComponentContext
-- get desktop (an XDesktop)
xdt = xsc~getDesktop
-- get current document
xdoc = xsc~getDocument
end
else
do
-- called from outside of OOo, create a connection
xcc = UNO.connect()
-- create a desktop service and its interface
service = "com.sun.star.frame.Desktop"
s_Desktop = xcc~getServiceManager~XMultiServiceFactory~createInstance(service)
xdt = s_Desktop~XDesktop
xdoc = xdt~getCurrentComponent()
end

xMCF = xcc~getServiceManager()

oText = xdoc~XTextDocument~getText()

xMSF = xdoc~XMultiServiceFactory

/*the database "addresses" with table "address" must be registered within OOo.org
 *fields: First Name, Last Name, Street, ZIP, City, Sex*/

/*create the masterfields and the textfields*/
dbfields = .array~of("First Name", "Last Name", "Street", "ZIP", "City")
masterfields = .array~new(dbfields~items)
fields = .array~new(dbfields~items)

do i=1 to dbfields~items
  /*master fields*/
  masterfields[i] = xMSF~createInstance("com.sun.star.text.FieldMaster.Database")
  /*only one out of DataBaseName, DataBaseURL, DataBaseResource can be set*/
  masterfields[i]~XPropertySet~setPropertyValue("DataBaseName","addresses")
  masterfields[i]~XPropertySet~setPropertyValue("DataTableName","address")
  masterfields[i]~XPropertySet~setPropertyValue("DataColumnName",dbfields[i])

  /*text fields*/
  fields[i] = xMSF~createInstance("com.sun.star.text.textfield.Database")
  fields[i]~XDependentTextField~attachTextFieldMaster(masterfields[i]~XPropertySet)
  fields[i]~XPropertySet~setPropertyValue("Content","<"||dbfields[i]||">")
```

```
end

/*set up the form letter address box*/
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("First Name",dbfields)]~
    XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("Last Name",dbfields)]~
    XTextContent,.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","LINE_BREAK"),.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("Street",dbfields)]~
    XTextContent,.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","LINE_BREAK"),.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("ZIP",dbfields)]~XTextContent,
    .false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),fields[getField("City",dbfields)]~XTextContent,
    .false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)
oText~insertControlCharacter(oText~XTextRange~getEnd(),bsf.getConstant("
    com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)

/*insert a conditional field*/
oCondField = xMSF~createInstance("com.sun.star.text.textfield.ConditionalText")
oCondField~XPropertySet~setPropertyValue("TrueContent","Miss")
oCondField~XPropertySet~setPropertyValue("FalseContent","Mister")
oCondField~XPropertySet~setPropertyValue("Condition","addresses.address.Sex==1")
oCondField~XPropertySet~setPropertyValue("CurrentPresentation","<SexConditional>")

/*last name field*/
oLNameField = xMSF~createInstance("com.sun.star.text.textfield.Database")
oLNameField~XDependentTextField~attachTextFieldMaster(masterfields[getField("Last Name",dbfields)
    ]~XPropertySet)
oLNameField~XPropertySet~setPropertyValue("Content","<"||dbfields[getField("Last Name",dbfields)
    ]||">")

oText~XSimpleText~insertString(oText~XTextRange~getEnd(),"Dear ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),oCondField~XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd()," ",.false)
oText~insertTextContent(oText~XTextRange~getEnd(),oLNameField~XTextContent,.false)
oText~XSimpleText~insertString(oText~XTextRange~getEnd(),"!",.false)

/*do the merger*/
/*document already saved?*/
path = xdoc~XModel~getURL()
if path=="" then
do
  oFilePicker = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")

  aListAny = bsf.createArray("Short.class",1)
  aListAny[1]=box("short",bsf.getStaticValue("com.sun.star.ui.dialogs.TemplateDescription","
      FILESAVE_AUTOEXTENSION"))
  oFilePicker~XInitialization~initialize(aListAny)
  xFileManager = oFilePicker~XFilterManager~appendFilter("Open Document Text","*.odt")

  file=.nil
  oFilePicker~XExecutableDialog~execute()
  file = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)

  if file==.nil then
  do
   .bsf.dialog~messageBox("No file selected - exiting".)
   exit
  end

  path=file
end
```

```
   .bsf.dialog~messageBox("here the error should occur")
106
   signal on ANY
108 /*save the document*/
   path=ConvertFromURL(path)
110 path=ConvertToURL(path)
   xdoc~XStorable~storeAsURL(path, .UNO~noprops)
112 signal off ANY

114 oMailMerge = xMCF~createInstanceWithContext("com.sun.star.text.MailMerge",xcc)
   oMailMerge~XPropertySet~setPropertyValue("DataSourceName","addresses")
116 oMailMerge~XPropertySet~setPropertyValue("DocumentURL",xdoc~XModel~getURL())

118 oMailMerge~XPropertySet~setPropertyValue("OutputType",box("SH",bsf.getStaticValue("
       com.sun.star.text.MailMergeType","FILE")))
   oMailMerge~XPropertySet~setPropertyValue("FileNamePrefix", "test_")
120 oMailMerge~XPropertySet~setPropertyValue("SaveAsSingleFile",box("boolean",.true))
   oMailMerge~XPropertySet~setPropertyValue("CommandType",box("I",bsf.getStaticValue("
       com.sun.star.sdb.CommandType","TABLE")))
122 oMailMerge~XPropertySet~setPropertyValue("Command","address")


124
   call bsf.import "com.sun.star.beans.NamedValue","NamedValue.class"
126 oMailMerge~XJob~execute(bsf.createArray(.NamedValue.class,0))
   exit 0
128
   ANY:
130   .bsf.dialog~messageBox("Could not delete file. Maybe opened? Exiting.")
     exit
132
   ::REQUIRES UNO.cls
134
   ::ROUTINE getField
136   USE ARG name, dbfields
     if dbfields~hasItem(name) then
138   do
       RETURN dbfields~index(name)
140   end
     else
142   do
       RETURN -1
144   end
```

Listing A.7: 11b_FormLetterGUI.rex. The form letter example including a graphical file selection dialog.

A.2.3. Supplement Example 15 – Alternative Way of Range Retrieval

Information on this listing can be found in section 4.1.3.

```
   s="Nonempty cells of" sRange .ENDOFLINE
2 do i=1 to oAddrs~items
     /*get one specific range*/
4    oAddr = oAddrs[i]
     /*iterate over each cell of the rectangular selection*/
6    nRowStart = oAddr~bsf.getFieldValue("StartRow")
     nRowEnd = oAddr~bsf.getFieldValue("EndRow")
8    nColStart = oAddr~bsf.getFieldValue("StartColumn")
     nColEnd = oAddr~bsf.getFieldValue("EndColumn")
10   do nRow = nRowStart to nRowEnd
       do nCol = nColStart to nColEnd
12     oCell = oSheet~XSheetCellRange~XCellRange~getCellByPosition(nCol,nRow)
       /*convert to human readable format*/
```

```
14      oCellAddress = oCell~XCellAddressable~getCellAddress()
        oConversion~XPropertySet~setPropertyValue("Address",oCellAddress)
16      s = s oConversion~XPropertySet~getPropertyValue("UserInterfaceRepresentation") .ENDOFLINE
        end
18    end
  end
```

Listing A.8: 15b_getRangeAddresses.rex. Alternative way of retrieving a range address.

A.2.4. Supplement Example 22 – Alternative Way of Conversion

Information on this listing can be found in section 5.1.1.

```
1 if oGraph~XServiceInfo~supportsService(s1) then
  do
3     /*convert a GraphicObjectShape*/
      oAnchor = oGraph~XTextContent~getAnchor()
5     oText = oAnchor~getText()
      oOldBitmap = oGraph~XPropertySet~getPropertyValue("Graphic")
7     oOldSize = oGraph~XShape~getSize()
      oOldPosition = oGraph~XShape~getPosition()
9
      oNewGraph = xMSF~createInstance(s1)
11    oNewGraph~XPropertySet~setPropertyValue("Graphic",oOldBitmap)
      oNewGraph~XShape~setSize(oOldSize)
13    oNewGraph~XShape~setPosition(oOldPosition)
      oText~insertTextContent(oAnchor, oNewGraph~XTextContent, .false)
15    oText~removeTextContent(oGraph~XTextContent)
  end
```

Listing A.9: 22b_ConvertLinkedGraphics.rex. Alternative conversion process for images of type `GraphicObjectShape`.

A.2.5. Supplement Example 23 – Graphic Size Graphic File Selection

Information on the listing can be found in section 2.2.2. The only addition when comparing to listing 5.3 provided with this listing is a graphic file selection dialog derived from Frysak [Frys08] and the Developer's Guide [Sun07, p. 1160].

```
  /*standard header*/
2 xsc = uno.getScriptContext()
  xdt = xsc~getDesktop()
4 xdoc = xsc~getDocument()
  xcc = xsc~getComponentContext()
6 xMCF = xcc~getServiceManager()
  xMSF = xdoc~XMultiServiceFactory
8
  call bsf.import "com.sun.star.beans.StringPair","StringPair.class"
10 filters=bsf.createArray(.StringPair.class,3)
  sp = .StringPair.class~new()
12 sp~bsf.setFieldValue("First","GIF")
  sp~bsf.setFieldValue("Second","*.gif")
14 filters~put(sp,1)
  sp = .StringPair.class~new()
16 sp~bsf.setFieldValue("First","JPG")
```

```
sp~bsf.setFieldValue("Second","*.jpg;*.jpeg")
filters~put(sp,2)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","PNG")
sp~bsf.setFieldValue("Second","*.png")
filters~put(sp,3)

sGraphicURL = .nil
do while sGraphicURL==.nil
oFilePicker=xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")
oFilePicker~XFilePicker~setMultiSelectionMode(.false)
oFilePicker~XFilterGroupManager~appendFilterGroup("Graphics",filters)
oFilePicker~XFilePicker~XExecutableDialog~execute()
sGraphicURL = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)
end

s = "com.sun.star.drawing.GraphicObjectShape"

if xdoc~uno.supportsService("TextDocument") then
  oDrawPage = xdoc~XDrawPageSupplier~getDrawPage()
else
  oDrawPage = xdoc~XModel~getCurrentController()~XDrawView~getCurrentPage()

oGraphic = xMSF~createInstance(s)
oGraphic~XPropertySet~setPropertyValue("GraphicURL",sGraphicURL)
oShape = oGraphic~XShape

oGraphicProvider = xMCF~createInstanceWithContext("com.sun.star.graphic.GraphicProvider",xcc)~
    XGraphicProvider
aMediaProperties = uno.CreateArray(.UNO~PROPERTYVALUE,1)
aMediaProperties[1] = uno.createProperty("URL",sGraphicURL)

/*obtain the graphic's property set*/
oGraphicPropertySet = oGraphicProvider~queryGraphicDescriptor(aMediaProperties)~XPropertySet

/*returns the "print size" set for a graphic in 100th mm*/
oGraphicSize = oGraphicPropertySet~getPropertyValue("Size100thMM")
iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height")
iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width")

/*are the sizes set?*/
/*if not, we set them at least in the right aspect ratio*/
if (iGraphicHeight==0 | iGraphicWidth==0) then
do
  oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
  iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height") *25
  iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width") *25

  /*ok, the preload mechanism failed - lets retrieve the graphic and then determine its size*/
  if (iGraphicHeight==0 | iGraphicWidth==0) then
  do
    oGraphic = oGraphicProvider~queryGraphic(aMediaProperties)
    if oGraphic <> .nil then
    do
    oGraphicPropertySet = oGraphic~XPropertySet
    oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
    iGraphicHeight = oGraphicSize~bsf.getFieldValue("Height") *25
    iGraphicWidth = oGraphicSize~bsf.getFieldValue("Width") *25
    end
    /*still nothing? -- well, giving up.*/
    else
    do
      .bsf.dialog~messageBox("Couldn't determine size. Exiting...")
      exit
    end
  end
end

/*set the new size and add the graphic to the draw page*/
```

```
size=.bsf~new("com.sun.star.awt.Size")
size~bsf.setFieldValue("height",iGraphicHeight)
size~bsf.setFieldValue("width",iGraphicWidth)

/*setSize sets the graphic size in 100th mm*/
oShape~setSize(size)
oDrawPage~XShapes~add(oShape~XShape)

::requires UNO.cls
```

Listing A.10: 23b_GraphicSizeGUI.rex. Graphic Size example with and graphic file selection dialog.

A.2.6. Supplement Example 24 – Export Graphics Graphic File Selection

Information on the listing can be found in section 2.2.2. The only addition when comparing to listing 5.5 provided with this listing is a graphic file selection dialog derived from Frysak [Frys08] and the Developer's Guide [Sun07, p. 1160].

```
/*standard header*/
xsc = uno.getScriptContext()
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()
xMSF = xdoc~XMultiServiceFactory

oFilePicker = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")

aListAny = bsf.createArray("Short.class",1)
aListAny[1]=box("short",bsf.getStaticValue("com.sun.star.ui.dialogs.TemplateDescription","
     FILESAVE_AUTOEXTENSION"))
oFilePicker~XInitialization~initialize(aListAny)
xFileManager = oFilePicker~XFilterManager~appendFilter("Portable Network Graphics","*.png")

file=.nil
oFilePicker~XExecutableDialog~execute()
file = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)

if file==.nil then
do
 .bsf.dialog~messageBox("No file selected - exiting".)
 exit
end

oSimpleFileAccess = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ucb.SimpleFileAccess")
if oSimpleFileAccess~XSimpleFileAccess2~exists(file) then
do
  signal on ANY
  oSimpleFileAccess~XSimpleFileAccess2~kill(file)
end


sFileURL=file

aFilterData = uno.createArray(.UNO~PROPERTYVALUE,5)
aFilterData[1]=uno.createProperty("PixelWidth",box("I",1000))
aFilterData[2]=uno.createProperty("PixelHeight",box("I",1000))
aFilterData[3]=uno.createProperty("LogicalWidth",box("I",1000))
aFilterData[4]=uno.createProperty("LogicalHeight",box("I",1000))
```

```
41 aFilterData[5]=uno.createProperty("Quality",box("I",90))

43 xView = xdoc~XModel~getCurrentController()
   xSelection = xView~XSelectionSupplier~getSelection()
45
   /*if we have a text document, we need to reduce the contained elements to XShapes*/
47 if xdoc~uno.supportsService("TextDocument") | xdoc~uno.supportsService("SpreadsheetDocument")
       then
   do
49   /*only shapes selected?*/
     if \ uno.getInterfaceNamesViaReflection(xSelection)~makeArray(" ")~hasItem("
         com.sun.star.drawing.XShapes") then
51   do
       .bsf.dialog~messageBox("No shapes selected. Exiting.")
53     exit
     end
55 end
   else
57 do
     /*xSelection contains the selected objects - if nothing is selected, the interface name is void
         */
59   if uno.getInterfaceName(xSelection)=="void" then
     do
61     xSelection = xView~XPropertySet~getPropertyValue("CurrentPage")
     end
63 end

65 xExporter = xMCF~createInstanceWithContext("com.sun.star.drawing.GraphicExportFilter",xcc)
   xExporter~XExporter~setSourceDocument(xSelection~XComponent)
67
   aArgs=uno.createArray(.UNO~PROPERTYVALUE,3)
69 aArgs[1]=uno.createProperty("MediaType","image/jpeg")
   aArgs[2]=uno.createProperty("URL",sFileURL)
71 aArgs[3]=uno.createProperty("FilterData",aFilterData)

73 a =xExporter~XFilter~filter(aArgs)
   .bsf.dialog~messageBox(a)
75
   exit 0
77
   ANY:
79   .bsf.dialog~messageBox("Could not delete file. Maybe opened? Exiting.")
     exit
81
   ::requires UNO.cls
```

Listing A.11: 24b_CombineAndExportGUI.rex. Combines and Exports using a GUI.

A.2.7. Supplement Example 25 – Combine and Export with GUI

In supplement to example 25, to following listing provides a GUI which allows to execute the script completely with the mouse. It includes a section for listing the graphic files to take into account, a path identifier to select the output file as well as the possibility to set the matrix dimensions according to the number of images specified. The basics of GUI creating are taken from Frysak [Frys08], the dialogs are partly derived from the Developer's Guide [Sun07, p. 1160].

```
1 /*standard header*/
  xsc = uno.getScriptContext()
```

```
xdt = xsc~getDesktop()
xdoc = xsc~getDocument()
xcc = xsc~getComponentContext()
xMCF = xcc~getServiceManager()

graphicholder = .array~new()

o_Dialog = xMCF~XMultiServiceFactory~createInstance("com.sun.star.awt.UnoControlDialogModel")
o_Dialog.Properties = o_Dialog~XPropertyset
o_Dialog.Properties~setPropertyValue("Width", box("int", 256))
o_Dialog.Properties~setPropertyValue("Height", box("int", 234))
o_Dialog.Properties~setPropertyValue("Title", "Graphics Selection")
o_Dialog.Factory = o_Dialog~XMultiServiceFactory
o_Dialog.Container = o_Dialog~XNameContainer
o_Dialog.Control = xMCF~XMultiServiceFactory~createInstance("com.sun.star.awt.UnoControlDialog")~
    XControl
o_Dialog.Control~setModel(o_Dialog~XControlModel)
o_Dialog.ControlContainer = o_Dialog.Control~XControlContainer
x_Toolkit = xMCF~XMultiServiceFactory~createInstance("com.sun.star.awt.Toolkit")~XToolkit
o_Dialog.Control~createPeer(x_Toolkit, .nil)
x_Frame = xMCF~XMultiServiceFactory~createInstance("com.sun.star.frame.Frame")~XFrame
x_Frame~initialize(o_Dialog.Control~XWindow)

oListBox = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlListBoxModel")
oListBox~XPropertySet~setPropertyValue("PositionX", box("int", 8))
oListBox~XPropertySet~setPropertyValue("PositionY", box("int", 8))
oListBox~XPropertySet~setPropertyValue("Width", box("int", (240)))
oListBox~XPropertySet~setPropertyValue("Height", box("int", 152))
oListBox~XPropertySet~setPropertyValue("ReadOnly", box("boolean",.true))
oListBox~XPropertySet~setPropertyValue("Name", "listGraphic")
listitems = uno.createArray("String.class",0)
oListBox~XPropertySet~setPropertyValue("StringItemList", listitems)
o_Dialog.Container~insertByName("list_listgraphic",oListBox)

oEditRowLabel = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlFixedTextModel")
oEditRowLabel~XPropertySet~setPropertyValue("PositionX", box("int", 13))
oEditRowLabel~XPropertySet~setPropertyValue("PositionY", box("int", 171))
oEditRowLabel~XPropertySet~setPropertyValue("Width", box("int", 90))
oEditRowLabel~XPropertySet~setPropertyValue("Height", box("int", 14))
oEditRowLabel~XPropertySet~setPropertyValue("Label", "Specify number of rows:")
o_Dialog.Container~insertByName("label_row", oEditRowLabel)

oEditRow = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlNumericFieldModel")
oEditRow~XPropertySet~setPropertyValue("PositionX", box("int", 95))
oEditRow~XPropertySet~setPropertyValue("PositionY", box("int", 168))
oEditRow~XPropertySet~setPropertyValue("Width", box("int", 24))
oEditRow~XPropertySet~setPropertyValue("Height", box("int", 14))
oEditRow~XPropertySet~setPropertyValue("DecimalAccuracy", box("short",0))
oEditRow~XPropertySet~setPropertyValue("StrictFormat", box("boolean",.true))
oEditRow~XPropertySet~setPropertyValue("Spin", box("boolean",.true))
oEditRow~XPropertySet~setPropertyValue("Value", box("double",1))
oEditRow~XPropertySet~setPropertyValue("Name",  "rowNumber")
o_Dialog.Container~insertByName("field_row", oEditRow)

oEditColumnLabel = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlFixedTextModel")
oEditColumnLabel~XPropertySet~setPropertyValue("PositionX", box("int", 137))
oEditColumnLabel~XPropertySet~setPropertyValue("PositionY", box("int", 171))
oEditColumnLabel~XPropertySet~setPropertyValue("Width", box("int", 90))
oEditColumnLabel~XPropertySet~setPropertyValue("Height", box("int", 14))
oEditColumnLabel~XPropertySet~setPropertyValue("Label", "Specify number of columns:")
o_Dialog.Container~insertByName("label_column", oEditColumnLabel)

oEditColumn = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlNumericFieldModel")
oEditColumn~XPropertySet~setPropertyValue("PositionX", box("int", 219))
oEditColumn~XPropertySet~setPropertyValue("PositionY", box("int", 168))
oEditColumn~XPropertySet~setPropertyValue("Width", box("int", 24))
oEditColumn~XPropertySet~setPropertyValue("Height", box("int", 14))
oEditColumn~XPropertySet~setPropertyValue("DecimalAccuracy", box("short",0))
oEditColumn~XPropertySet~setPropertyValue("StrictFormat", box("boolean",.true))
```

```
oEditColumn~XPropertySet~setPropertyValue("Spin", box("boolean",.true))
oEditColumn~XPropertySet~setPropertyValue("Value", box("double",1))
oEditColumn~XPropertySet~setPropertyValue("Name",  "columnNumber")
o_Dialog.Container~insertByName("field_column", oEditColumn)

oEditPathLabel = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlFixedTextModel")
oEditPathLabel~XPropertySet~setPropertyValue("PositionX", box("int", 13))
oEditPathLabel~XPropertySet~setPropertyValue("PositionY", box("int", 193))
oEditPathLabel~XPropertySet~setPropertyValue("Width", box("int", 60))
oEditPathLabel~XPropertySet~setPropertyValue("Height", box("int", 14))
oEditPathLabel~XPropertySet~setPropertyValue("Label", "Specify output path:")
o_Dialog.Container~insertByName("label_path", oEditPathLabel)

oPathEdit = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlEditModel")
oPathEdit~XPropertySet~setPropertyValue("PositionX", box("int", 81))
oPathEdit~XPropertySet~setPropertyValue("PositionY", box("int", 190))
oPathEdit~XPropertySet~setPropertyValue("Width", box("int", 96))
oPathEdit~XPropertySet~setPropertyValue("Height", box("int", 14))
oPathEdit~XPropertySet~setPropertyValue("ReadOnly", box("boolean", .true))
o_Dialog.Container~insertByName("edit_path", oPathEdit)

oPathButton = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlButtonModel")
oPathButton~XPropertySet~setPropertyValue("PositionX", box("int", 185))
oPathButton~XPropertySet~setPropertyValue("PositionY", box("int", 190))
oPathButton~XPropertySet~setPropertyValue("Width", box("int", 58))
oPathButton~XPropertySet~setPropertyValue("Height", box("int", 14))
oPathButton~XPropertySet~setPropertyValue("Name", "select")
oPathButton~XPropertySet~setPropertyValue("Label", "Select...")
o_Dialog.Container~insertByName("button_path", oPathButton)
oPathButton_Control = o_Dialog.ControlContainer~getControl("button_path")
oPathButton_Control~XButton~bsf.addEventListener("action", "actionPerformed","select")


o_Button1 = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlButtonModel")
o_Button1.Properties = o_Button1~XPropertySet
o_Button1.Properties~setPropertyValue("PositionX", box("int", 8))
o_Button1.Properties~setPropertyValue("PositionY", box("int", 212))
o_Button1.Properties~setPropertyValue("Width", box("int", 116))
o_Button1.Properties~setPropertyValue("Height", box("int", 14))
o_Button1.Properties~setPropertyValue("Name", "addGraphic")
o_Button1.Properties~setPropertyValue("Label", "Add a graphic...")
o_Dialog.Container~insertByName("button_addgraphic", o_Button1)
o_Button1.Control = o_Dialog.ControlContainer~getControl("button_addgraphic")
o_Button1.Control~XButton~bsf.addEventListener("action", "actionPerformed","addGraphic")
o_Dialog.Control~XWindow~setVisible(.true)


oButton2 = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlButtonModel")
oButton2~XPropertySet~setPropertyValue("PositionX", box("int", 132))
oButton2~XPropertySet~setPropertyValue("PositionY", box("int", 212))
oButton2~XPropertySet~setPropertyValue("Width", box("int", 116))
oButton2~XPropertySet~setPropertyValue("Height", box("int", 14))
oButton2~XPropertySet~setPropertyValue("Name", "execute")
oButton2~XPropertySet~setPropertyValue("Label", "Execute!")
o_Dialog.Container~insertByName("button_execute", oButton2)
oButton2_Control = o_Dialog.ControlContainer~getControl("button_execute")
oButton2_Control~XButton~bsf.addEventListener("action", "actionPerformed","execute")
oButton2_Control~XButton~bsf.addEventListener("action", "disposing", "close")
o_Dialog.Control~XWindow~setVisible(.true)

do forever
  eventText=bsf.pollEventText(1000) -- timeout of 100/1000 sec
  if eventText="execute" then
  do
  oRectangle = .bsf~new("com.sun.star.awt.Rectangle")
  if checkMatrix(o_Dialog.ControlContainer,graphicholder) then
  do
    path = o_Dialog.ControlContainer~getControl("edit_path")~getModel()~XPropertySet~
         getPropertyValue("Text")
```

```
      row = o_Dialog.ControlContainer~getControl("field_row")~getModel()~XPropertySet~
          getPropertyValue("Value")
      column = o_Dialog.ControlContainer~getControl("field_column")~getModel()~XPropertySet~
          getPropertyValue("Value")

      oButton2~XPropertySet~setPropertyValue("Enabled",box("boolean",.false))

      ret = executeCombine(graphicholder,path,row,column,xdoc,xMCF,xcc)
      if ret then
      do
        oMsgBox = x_Toolkit~XMessageBoxFactory~createMessageBox(o_Dialog.Control~getPeer(),
            oRectangle,"infobox",1,"Saving","Saving successful.")
        oMsgBox~execute()
        --leave
      end
      else
      do
        oMsgBox = x_Toolkit~XMessageBoxFactory~createMessageBox(o_Dialog.Control~getPeer(),
            oRectangle,"errorbox",1,"Saving","Errors occurred. Not saved.")
        oMsgBox~execute()
      end
      oButton2~XPropertySet~setPropertyValue("Enabled",box("boolean",.true))
    end
    else
    do
      oMsgBox = x_Toolkit~XMessageBoxFactory~createMessageBox(o_Dialog.Control~getPeer(),oRectangle
          ,"errorbox",1,"Matrix","Matrix dimensions don't match picture numbers. Please correct.")
      oMsgBox~execute()
    end
    end
    else if eventText="addGraphic" then
    graphicholder = addGraphic(o_Dialog.ControlContainer,graphicholder,xMCF)
    else if eventText="close" then
    leave
    else if eventText="select" then
      call selectPath o_Dialog.ControlContainer,xMCF
  end

  o_Dialog.Control~XComponent~dispose()

  ::requires UNO.cls

  ::routine selectPath
  use arg o_Dialog.ControlContainer xMCF
  oFilePicker = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")

  aListAny = bsf.createArray("Short.class",1)
  aListAny[1]=box("short",bsf.getStaticValue("com.sun.star.ui.dialogs.TemplateDescription","
      FILESAVE_AUTOEXTENSION"))
  oFilePicker~XInitialization~initialize(aListAny)
  xFileManager = oFilePicker~XFilterManager~appendFilter("JPEG File Interchange Format","*.jpg")

  file=.nil
  oFilePicker~XExecutableDialog~execute()
  file = oFilePicker~XFilePicker~getFiles()~makeArray(" ")~at(1)


  sDir = file
  if \ (sDir~length==0) then
  do
    oPathEdit=o_Dialog.ControlContainer~getControl("edit_path")~getModel
    oPathEdit~XPropertySet~setPropertyValue("Text",sDir)
    oSimpleFileAccess = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ucb.SimpleFileAccess
        ")
    if oSimpleFileAccess~XSimpleFileAccess2~exists(file) then
    do
      signal on ANY
      oSimpleFileAccess~XSimpleFileAccess2~kill(file)
      signal off ANY
```

```
  end
end
return .true

ANY:
  .bsf.dialog~messageBox("Could not delete file. Maybe opened? Exiting.")
  return .false


::routine checkMatrix
use arg o_Dialog.ControlContainer graphicholder

oEditRow = o_Dialog.ControlContainer~getControl("field_row")~getModel()
iRows = oEditRow~XPropertySet~getPropertyValue("Value")
oEditColumn = o_Dialog.ControlContainer~getControl("field_column")~getModel()
iCols = oEditColumn~XPropertySet~getPropertyValue("Value")

iAmount = graphicholder~items

if iAmount = iCols*iRows then
  ret= .true
else
  ret= .false

return ret

::routine addGraphic
use arg o_Dialog.ControlContainer graphicholder xMCF

call bsf.import "com.sun.star.beans.StringPair","StringPair.class"
filters=bsf.createArray(.StringPair.class,3)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","GIF")
sp~bsf.setFieldValue("Second","*.gif")
filters~put(sp,1)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","JPG")
sp~bsf.setFieldValue("Second","*.jpg;*.jpeg")
filters~put(sp,2)
sp = .StringPair.class~new()
sp~bsf.setFieldValue("First","PNG")
sp~bsf.setFieldValue("Second","*.png")
filters~put(sp,3)

oFilePicker = xMCF~XMultiServiceFactory~createInstance("com.sun.star.ui.dialogs.FilePicker")
oFilePicker~XFilePicker~setMultiSelectionMode(.true)
oFilePicker~XFilterGroupManager~appendFilterGroup("Graphics",filters)
oFilePicker~XFilePicker~XExecutableDialog~execute()
aFiles = oFilePicker~XFilePicker~getFiles()~makeArray(" ")

if aFiles~items < 2 then
do
  if aFiles~items==1 then
    graphicholder~append(aFiles~at(1))
  else
    return graphicholder
end
else
do
  do i=2 to aFiles~items
    graphicholder~append(aFiles~at(1)||"/"||aFiles~at(i))
  end
end

oListBox=o_Dialog.ControlContainer~getControl("list_listgraphic")~getModel()
listitems = uno.createArray("String.class",graphicholder~items)
do i=1 to graphicholder~items
  listitems~put(graphicholder~at(i),i)
end
```

```
oListBox~XPropertySet~setPropertyValue("StringItemList",listitems)
return graphicholder

::routine executeCombine
use arg imports path raster_y raster_x xdoc xMCF xcc

xMSF = xdoc~XMultiServiceFactory

dpi = 300
exportfile = path

dp100thmm = dpi/(2.54*10*100)

/*create a new drawpage*/
oDrawPages = xdoc~XDrawPagesSupplier~getDrawPages()
iDrawPageCount = oDrawPages~XIndexAccess~getCount()
oDrawPages~insertNewByIndex(iDrawPageCount)
oDrawPage = oDrawPages~XIndexAccess~getByIndex(iDrawPageCount) --XDrawPage

/*set pictures at the appropriate positions*/
oGraphicProvider = xMCF~createInstanceWithContext("com.sun.star.graphic.GraphicProvider",xcc)~
     XGraphicProvider
pixelwidths = .array~new
pixelheights = .array~new
heightmax = 0
widthmax = 0

do i=1 to imports~items
  sGraphicURL = imports~at(i)
  aMediaProperties = uno.CreateArray(.UNO~PROPERTYVALUE,1)
  aMediaProperties[1] = uno.createProperty("URL",sGraphicURL)
  oGraphicPropertySet = oGraphicProvider~queryGraphicDescriptor(aMediaProperties)~XPropertySet

  /*obtain the pixel sizes*/
  oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
  pixelwidths[i] = oGraphicSize~bsf.getFieldValue("Width")
  pixelheights[i] = oGraphicSize~bsf.getFieldValue("Height")

  /*no size contained in SizePixel property?*/
  if pixelwidths[i]==0 | pixelheights[i]==0 then
  do
  oGraphic = oGraphicProvider~queryGraphic(aMediaProperties)
    if oGraphic <> .nil then
    do
    oGraphicPropertySet = oGraphic~XPropertySet
    oGraphicSize = oGraphicPropertySet~getPropertyValue("SizePixel")
    pixelheights[i] = oGraphicSize~bsf.getFieldValue("Height")
    pixelwidths[i] = oGraphicSize~bsf.getFieldValue("Width")
    end
    /*still nothing? -- well, giving up.*/
    else
    do
      .bsf.dialog~messageBox("Couldn't determine graphic object size.")
      return 0
    end
  end

  if pixelwidths[i]>widthmax then
    widthmax = pixelwidths[i]

  if pixelheights[i]>heightmax then
    heightmax = pixelheights[i]

end

/*calculate the raster size*/
rasterwidth = widthmax/dp100thmm
rasterheight = heightmax/dp100thmm
```

```
   /*we need a background shape*/
339 oBackground = xMSF~createInstance("com.sun.star.drawing.RectangleShape")
   oBackground~XPropertySet~setPropertyValue("LineStyle",bsf.getStaticValue("
        com.sun.star.drawing.LineStyle","NONE"))
341 oBackground~XPropertySet~setPropertyValue("FillStyle",bsf.getStaticValue("
        com.sun.star.drawing.FillStyle","NONE"))

343 /*set background size*/
   width = rasterwidth*raster_x
345 height = rasterheight*raster_y
   oBackgroundSize = .bsf~new("com.sun.star.awt.Size")
347 oBackgroundSize~bsf.setFieldValue("Height",trunc(height))
   oBackgroundSize~bsf.setFieldValue("Width",trunc(width))
349 oBackground~XShape~setSize(oBackgroundSize)

351 /*set background position*/
   oBackgroundPos=.bsf~new("com.sun.star.awt.Point")
353 oBackgroundPos~bsf.setFieldValue("X",0)
   oBackgroundPos~bsf.setFieldValue("Y",0)
355 oBackground~XShape~setPosition(oBackgroundPos)
   oDrawPage~XShapes~add(oBackground~XShape)
357
   /*iterate and set positions*/
359 pageoffsetheight = 0
   do j=1 to raster_y
361   pageoffsetwidth = 0
     do k=1 to raster_x
363     i=(j-1)*raster_x+k
       sGraphicURL=imports[i]
365     oGraphic = xMSF~createInstance("com.sun.star.drawing.GraphicObjectShape")
       oGraphic~XPropertySet~setPropertyValue("GraphicURL",sGraphicURL)
367     oShape = oGraphic~XShape

369     height = pixelheights[i]/dp100thmm
       width = pixelwidths[i]/dp100thmm
371
       /*set the new size*/
373     size=.bsf~new("com.sun.star.awt.Size")
       size~bsf.setFieldValue("height",trunc(height))
375     size~bsf.setFieldValue("width",trunc(width))
       oShape~setSize(size)
377
       /*calculate offset*/
379
       customoffsetwidth = pageoffsetwidth + ((rasterwidth - width)/2)
381    customoffsetheight = pageoffsetheight + ((rasterheight - height)/2)

383    /*set the new position*/
       pos=.bsf~new("com.sun.star.awt.Point")
385    pos~bsf.setFieldValue("X",trunc(customoffsetwidth))
       pos~bsf.setFieldValue("Y",trunc(customoffsetheight))
387    oShape~setPosition(pos)

389    oDrawPage~XShapes~add(oShape)

391    pageoffsetwidth = pageoffsetwidth + rasterwidth
     end
393  pageoffsetheight = pageoffsetheight + rasterheight
   end
395
   /*export to one image file*/
397 xSelectionSupplier = xdoc~XModel~getCurrentController~XSelectionSupplier
   /*select all XShapes of the current page - we want to export all of them*/
399 xSelectionSupplier~select(oDrawPage~XShapes)

401 /*get the selection*/
   xSelection = xSelectionSupplier~getSelection()
403
   /*calculate the size*/
```

```
size_x = trunc(pageoffsetwidth*dp100thmm)
size_y = trunc(pageoffsetheight*dp100thmm)

/*setup the filter*/
aFilterData = uno.createArray(.UNO~PROPERTYVALUE,5)
aFilterData[1]=uno.createProperty("PixelWidth",box("I",size_x))
aFilterData[2]=uno.createProperty("PixelHeight",box("I",size_y))
aFilterData[3]=uno.createProperty("LogicalWidth",box("I",trunc(pageoffsetwidth)))
aFilterData[4]=uno.createProperty("LogicalHeight",box("I",trunc(pageoffsetheight)))
aFilterData[5]=uno.createProperty("Quality",box("I",90))

/*get the GraphicExportFilter*/
xExporter = xMCF~createInstanceWithContext("com.sun.star.drawing.GraphicExportFilter",xcc)
xExporter~XExporter~setSourceDocument(xSelection~XComponent)

aArgs=uno.createArray(.UNO~PROPERTYVALUE,3)
aArgs[1]=uno.createProperty("MediaType","image/jpeg")
aArgs[2]=uno.createProperty("URL",exportfile)
aArgs[3]=uno.createProperty("FilterData",aFilterData)

ret = xExporter~XFilter~filter(aArgs)
return ret
```

Listing A.12: 25b_CombineAndExportGUI.rex. Combines and Exports using a GUI.

B. Gantt Project Charts

B.1. Proposed Chart at Kick-Off Meeting

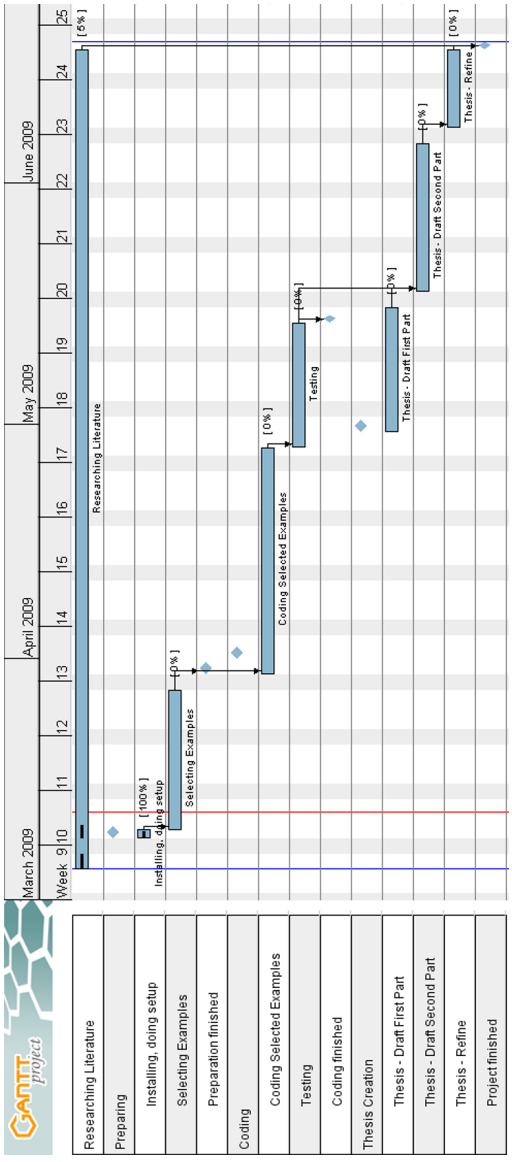


Figure B.1.: The initial Gantt chart of the project, created with GanttProject 2.0.9, `http://www.ganttproject.biz`

B.2. Final Chart

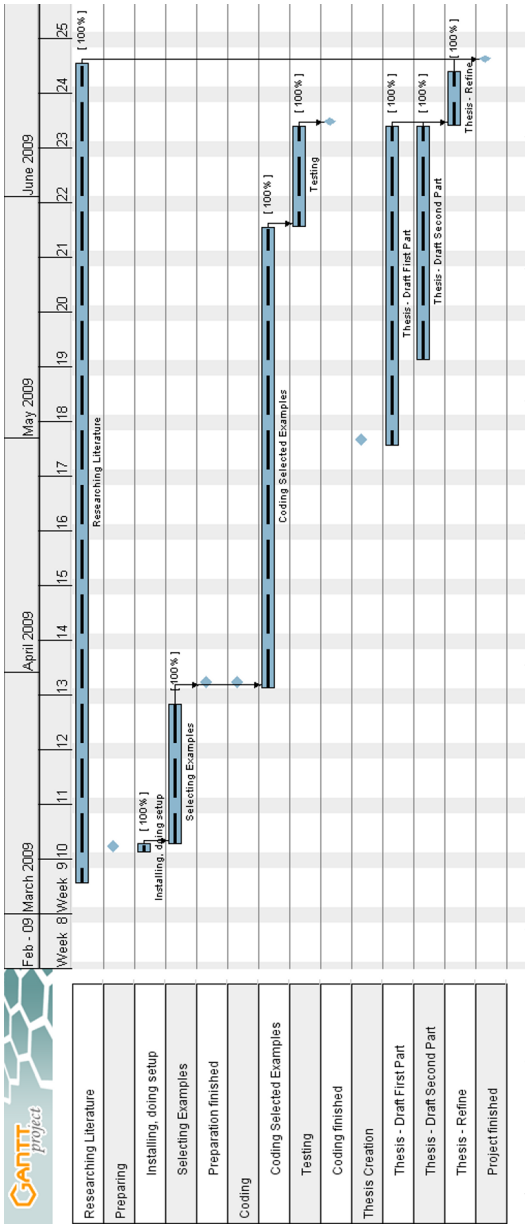


Figure B.2.: The final Gantt chart of the project, created with GanttProject 2.0.9, `http://www.ganttproject.biz`

Bibliography

[Aham05] *Ahammer, Andreas*: OpenOffice.org Automation: Object Model, Scripting Languages, "Nutshell"-Examples. Bachelor course paper, Vienna University of Economics and Business Administration, 2005.

[Apac] *Apache Jakarta Project*: Bean Scripting Framework. http://jakarta.apache.org/bsf/.

[Burg06] *Burger, Martin*: OpenOffice.org Automatisation with Object Rexx. Bachelor course paper, Vienna University of Economics and Business Administration, May 2006.

[Flat] *Flatscher, Rony G.*: Automatisierung von Java Anwendungen (10). Slideset for Lecture "Automatisierung von Java Anwendungen". http://wi.wu-wien.ac.at/rgf/wu/lehre/autojava/material/foils/AutoJava_10_BSF4Rexx_4_OOo.pdf, retrieved on Jun 10, 2009.

[Flat01] *Flatscher, Rony G.*: Java Bean Scripting with Rexx. In: 12™ International Rexx Symposium, Raleigh, North Carolina, USA 2001, retrieved from http://wi.wu-wien.ac.at/rgf/rexx/orx12/JavaBeanScriptingWithRexx_orx12.pdf on Mar 19, 2009.

[Flat05a] *Flatscher, Rony G.*: Automating OpenOffice.org with ooRexx: Architecture, Gluing to Rexx Using BSF4REXX. In: The 2005 International Rexx Symposium, Los Angeles, California, USA 2005, retrieved from http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf on Mar 19, 2009.

[Flat05b] *Flatscher, Rony G.*: Automating OpenOffice.org with ooRexx: ooRexx Nutshell Examples for Write and Calc. In: The 2005 International Rexx Symposium, Los Angeles, California, USA 2005, retrieved from http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Nutshell_OOo.pdf on Mar 19, 2009.

[Flat06] *Flatscher, Rony G.*: UNO.CLS: An (Open) Object Rexx Module for Universal Network Objects. In: 2006 International Rexx Symposium, Austin, Texas, USA 2006,

retrieved from http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_UNO.pdf on Apr 15, 2009.

[Flat08a] *Flatscher, Rony G.*: BSF4REXX. http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/, Sep 2008, retrieved on Mar 14, 2009.

[Flat08b] *Flatscher, Rony G.*: refcardOOo. Wirtschaftsuniversität Wien, 2008. edition, Sep 2008, retrieved from http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/refcardOOo.pdf on Mar 19, 2009.

[Foru04] *Forum User "DannyB"*: How to insert a graphic into a document. Forum thread, http://www.oooforum.org/forum/viewtopic.phtml?t=14979, Dec 2004, retrieved on May 26, 2009.

[Free07] *Free Software Foundation, Inc.*: GNU Lesser General Public License Version 3. http://www.gnu.org/licenses/lgpl-3.0.html, Jun 2007, retrieved on Jun 09, 2009.

[Frys08] *Frysak, Josef*: Automating Open Office – ooRexx Nutshells. Bachelor course paper, Vienna University of Economics and Business Administration, 2008.

[Gmei08] *Gmeiner, Michael*: OpenOffice.org: Selected Pitonyak' Nutshells in ooRexx. Seminar course paper, Vienna University of Economics and Business Administration, 2008.

[Gund07] *Gundacker, Dominik*: Automated Creation of Guideposts & Hints for Presenters in OpenOffice.org Impress. Bachelor course paper, Vienna University of Economics and Business Administration, 2007.

[Hinz06] *Hinz, Michael*: OpenOffice.org Calc Automation Using ooRexx. Bachelor course paper, Vienna University of Economics and Business Administration, 2006.

[Opena] *OpenOffice.org Project*: Application Framework Project. http://framework.openoffice.org/scripting/, retrieved on Jun 09, 2009.

[Openb] *OpenOffice.org Project*: OpenOffice.org API Documentation. retrieved from http://api.openoffice.org on Jun 17, 2009.

[Openc] *OpenOffice.org Project*: OpenOffice.org the free and open productivity suite. http://www.openoffice.org/, retrieved on Jun 09, 2009.

[Opend] *OpenOffice.org Project*: Why OpenOffice.org. http://why.openoffice.org/, retrieved on Jun 09, 2009.

[Open03] *OpenOffice.org Quality Assurance*: Issue 10933: Data sorting by using more than 3 items (columns). http://qa.openoffice.org/issues/show_bug.cgi?id=10933, Jan 2003, retrieved on May 29, 2009.

[Open05] *OpenOffice.org Quality Assurance*: Issue 44828: tools-detective-trace for cross-sheet formulae. http://qa.openoffice.org/issues/show_bug.cgi?id=44828, Mar 2005, retrieved on May 29, 2009.

[Pito04] *Pitonyak, Andrew Douglas*: OpenOffice.org Macros Explained. Hentzenwerke Publishing 2004.

[Pito07] *Pitonyak, Andrew*: Useful Macro Information For OpenOffice. 1012. edition, Jul 2007, retrieved from http://www.pitonyak.org/AndrewFontMacro.odt on Mar 15, 2009.

[Pito09] *Pitonyak, Andrew*: OpenOffice.org Base Macro Programming By Andrew Pitonyak. 35. edition, 2009, retrieved from http://www.pitonyak.org/database/AndrewBase.odt on Apr 22, 2009.

[Prem06] *Prem, Matthias*: ooRexx Snippets for OpenOffice.org Writer. Bachelor course paper, Vienna University of Economics and Business Administration, Jul 2006.

[Rexx] *Rexx Language Association*: Open Object Rexx. http://www.oorexx.org, retrieved on Mar 14, 2009.

[Rexx07] *Rexx Language Association*: Open Object Rexx: Rapid Application Development for Today's Business Environment. http://www.oorexx.org/ooRexx-Brochure.pdf, 2007, retrieved on Jun 09, 2009.

[SaJS04] *Samyn, Olivier; Jacobi, Sven; andTom Schindl, Michael Hoennig*: OOSnippets: Export JPEG. http://codesnippets.services.openoffice.org/Office/Office.GraphicExport.snip, Jun 2004, retrieved from http://codesnippets.services.openoffice.org on May 26, 2009.

[Schm07] *Schmid, Stefan*: Facilitate Data Access in OpenOffice.org using ooRexx. Bachelor course paper, Vienna University of Economics and Business Administration, Feb 2007.

[Scho07] *Scholz, Nicole*: Open Office Draw Nutshells. Seminar course paper, Vienna University of Economics and Business Administration, May 2007.

[Scho09] *Scholz, Nicole*: Open Office API-Viewer. Master thesis, Vienna University of Economics and Business Administration, Feb 2009.

[SpRG06] *Spanberger, David; Realfsen, Åsmund; Görlich, Gerhard*: BSF4Rexx and OpenOffice.org Nutshell-Examples. Seminar course paper, Vienna University of Economics and Business Administration, 2006.

[Sun07] Sun Microsystems, Inc., OpenOffice.org 2.3 Developer's Guide. Jun 2007, retrieved from http://wiki.services.openoffice.org/w/index.php?title=Special:Collection/render_article/&arttitle=Documentation/DevGuide/OpenOffice.org_Developers_Guide&writer=rl on Mar 19, 2009.

[ubun] *ubuntuusers*: Wiki / OpenOffice.org/Installation. http://wiki.ubuntuusers.de/OpenOffice.org/Installation, retrieved on Jun 09, 2009. In German.

[Wiki] *Wikimedia Foundation, Inc.*: Wikipedia – The Free Encyclopedia. http://en.wikipedia.org/wiki/, retrieved on Jun 09, 2009.