WIRTSCHAFTSUNIVERSITÄT WIEN

BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:

Datenverarbeitungs Automatisierung mit OpenOffice.org

Englischer Titel der Bakkalaureatsarbeit:

Flexible Word Processing Automation with OpenOffice.org

Verfasserin/Verfasser:	**Kauril Michael**
Matrikel-Nr.:	**0251728**
Studienrichtung:	**J033 526 Bakkalaureat Wirtschaftsinformatik**
Kurs:	**1526 Vertiefungskurs VI / Bakkalaureatsarbeit – Electronic Commerce**
Textsprache:	**Englisch**
Betreuerin/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

Table of Contents

List of Figures

List of tables

1 Introduction

The work is carried out in cooperation with the MA 14, responsible for the whole Information and Communication Technology in Vienna, and the MA 22, responsible for environment protection. The reason for this project is that the city of Vienna is trying to switch from commercial software to open source software. To increase the accomplishment of this changeover the MA 14 is trying to create automation processes in the new open source environment to decrease the number of manual processes and to improve the convenience of the new system. This automation is carried out within the context of OpenOffice.org, when talking about this open source office suite we are refering to it also as OOo.

1.1 Abstract

The work gives a detailed description of the realisation of the automation project of open source office software conducted in cooperation with the MA 14 and MA 22. It provides a theoretical background in order to get a better understanding of the implementation part of the work. The chapters concerned with theoretic concepts describe the design of OpenOffice.org and gives information about the API of OOo and the used programming language. It also gives information about the system requirements concerning installed software. The practical part describes in detail the realisation and gives explanations to every part of the programming code.

1.2 Project Definition

The main focus of these bachelor thesis is the automation of template handling in Open Office, accomplished with the use of a scripting language. The aim of the project is to produce a macro which automates the template handling of the MA 22. The MA 22 has many manual processes concerning the office staff, especially considering correspondency. Within this process the clerk has to search for the current template, open, edit, and finally save and print it. The problem which arises in this matter is that the templates, which are used to provide a cooperative identity, are changing on a regulary base. On this account there exists the risk to pick a wrong and outdated template. The whole process is very slow because it is done manually and carried out very often.

Furthermore there is a high usage of memory space, because every document is saved seperately as “PDF” and “Open Office Document”. To solve all these problems the macro should be able to pick the right template, load already finished documents, save content seperately from the template, export the document in “PDF” and print it. Regarding the realisation the MA gave no boundaries how and in which programming language to dissolve the problem. By reason of this fact the first step which was done was to get an overview of the design of Open Office and choose the appropriate scripting language.

2 System Requirements

This main part of the work provides a theoretic overview of the technical requirements to the system and about the concepts which lie behind the used technologies for the realisation.

2.1 OpenOffice.org

Office applications are one of the most widespread applications, the reason for this is that they are very multifunctional and that everyone has to use them for work-related or personal stuff. Such programs offer the user a wide range of features for instance to write letters, to conduct calculations in spreadsheets, to make multimedia presentations or to create a database. On the market there are a high number of products offered, like MS-Office, Star Office, OpenOffice.org or WordPerfect Office, the most well known is Microsofts Office Suite which has become a “quasi standard”.
[OPEN06]

2.1.1 History

The sourcecode of OpenOffice.org is based on the StarOffice suite which was produced by StarDivision, a software firm founded in Germany in the mid 1980s. 1999 Sun Microsystem took over the company and made the source code of OpenOffice.org, which was a new version of Star Office, available in 2000. Since this time the office suite is

developed continously and its popularity is rising. This can be seen by the many linux distributions which integrate OpenOffice.org within their installation and many public institutions who are migrating from proprietary office applications, for example the “LiMux” project of the city of munich or the “Wienux” project of the city of vienna. Another interesting development in this area is that google and sun microsystems joined a strategic partnership in 2005 to support the spread of OpenOffice.org.
[OPEN06]

2.1.2 Overview

OpenOffice.org is an open source office suite by Sun Microsystems that is freely available under the GNU Lesser General Puplic License (LGPL). The current release 2.1 is currently downloadable on the Webpage “http://download.openoffice.org/index.html”. The mission statement gives an very good idea what the whole project is about and what idea lies behind it:

"To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format." [OPEN06]

The advantages of this open source project are that it is an international office suite which is available on all major operating systems, therefore it is vendor-neutral and can make usage of next-stage architecture. Furthermore all functionalities and data can be accessed using the APIs, which are open component based, and a file format that is vendor-neutral by the use of XML.

The office suite consists of six different components:

- Writer application

  This application is a fully equipped word processor that has every feature a writing tool needs like for the creation of professional documents, memos or even booklets.

- Calc application

  This is the spreadsheet programm for the professional working with data. It can be used for tasks like calculation, analysation or presentation.

- Math application

  The Math component is an equation editor for text documents but can also be used stand-alone.

- Draw application

  Draw is an application which gives the user tools to edit graphics and diagrams.

- Impress application

  Impress is a tool that can be used to create multimedia presentations.

- Base application

  This component offers the ability to manipulate databases in OpenOffice.org.

[OPEN06]

2.1.3 Automation

The process of automation means that a small programm fullfills a set of tasks, very often linked together, in a minimum amount of time by taking control of other programs and applications. Such a program is normally called “macro”. In our context the process of automation means to take control over one application of the office suite by using a programming language.

“A Macro is used to automate task in OpenOffice.org. A macro can automate actions that otherwise require exhausting error-prone manual labor. Currently, the automatic actions are most easily created by writing Macros in Ooo Basic. The new scripting framework in OOo version 2 should ease the use of other languages, but Basic is still the most easiest to use. ...” [PITO06]

2.1.4 Implementation Languages

There are different ways and possible programming languages which can be used to automate OpenOffice. Except the integrated scritpting language all other languages are using the UNO technology, which is an interface to OOo and will be discussed later.

- OpenOffice.org Basic (also called Star Basic)
  OOo Basic is a macro language which is very similar to Visual Basic and is included in OpenOffice.org. It cooperates with the UNO interface so that it is possible to write UNO programs directly inside the office suite. Even so this language is very easy to use and it is very powerful because of its implemented features. Compared to the other possibilities of automating Ooo this scripting language has less features and isn't capable to communicate with other languages. Supplementary it has to be started from inside of OpenOffice.org and isn't executeable from the shell.

- Java
  This programming language is very powerful esspecially compared to scripting languages. The API of OpenOffice.org includes an interface so called “Java UNO” which offers the user the ability to use Java to automate Ooo, displayed in figure 1. This gives programmers the possibility to create and implement UNO components completley in Java, as well as the use of the Java standard library with a large set of features.

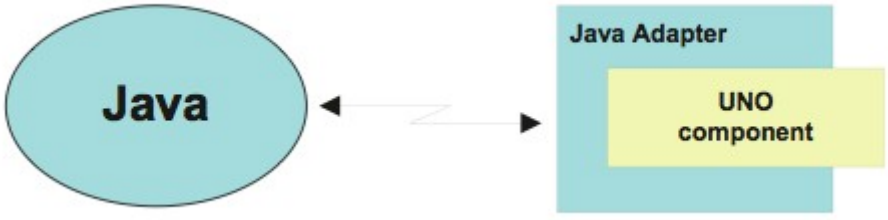

Figure 1: Java UNO Support [FLAT05]

- C++

  C++ like Java has a huge API to script the open source office suite although it is not very easy to use and has a higher level of complexity. Because OpenOffice.org was developed in C++ it offers the fastest way to communicate with the application.

- Object Rexx

  Object Rexx is a very easy to use scripting language which is even capable of taking advantage of the Java API by using BSF4Rexx. Therefore the user is able to automate OpenOffice.org with an easy to use scripting language and the use of Java features.

[OPEN06]

2.1.5 API

The API of OpenOffice.org was made to support any common programming language. The reason behind this approach is to make the internal implementation exchangeable so that it is easy for developers to extend the functionalities by new solutions or features, without being bound to any specific language. To reach this target OpenOffice.org was split into a lot of components which are combined to deliver their functionality. The concept of working with components simplifies the working with the API and make them more manageable because they are like building blocks that interact with each other to offer high level functionalities. With this concepts it is easy to exchange them with different implementations that provide similar functionality. The description of the interaction of the components will be given in one of the next chapters. The API reference make use of so called UNOIDL data types which are specific for the OpenOffice.org API and are needed to map to the different types of any language that is supported.

[OPEN05]

2.2 Architecture of OOo

2.2.1.1 UNO Concept

The Universal Network Object of OpenOffice.org is a powerful interface to get a connection by the use of many programming languages. The Developers Guide of OOo gives a very good description of UNO:

“UNO (pronounced ['ju:nou]) stands for Universal Network Objects and is the base component technology for OpenOffice.org. You can utilize and write components that interact across languages, component technologies, computer platforms, and networks.”
[OPEN05]

UNO is avaible like OpenOffice.org for many platforms like Linux, Windows or Mac OS X, so it is platform independent. It supports the programming languages Java, C++ and Basic. In addition it can be accessed through Microsofts COM technology by other languages and has a language binding for Python. Versions of OpenOffice.org higher than 2.0 are able to be programmed with .NET languages by using the Common Language Infrastructure binding. Furthermore the new scripting framework makes it possible to use the API through scripting languages, like Beanshell, Javascript or Jython. The concept of UNO is client/server based, therefore there has to be a connection between the server and the client via TCP/IP to be initialised. This gives the opportunity that the server and the client don't need to be on the same computer. For example they could also establish a connection via the internet for example. The communication between client and server is described in figure 2:
[OPEN05]

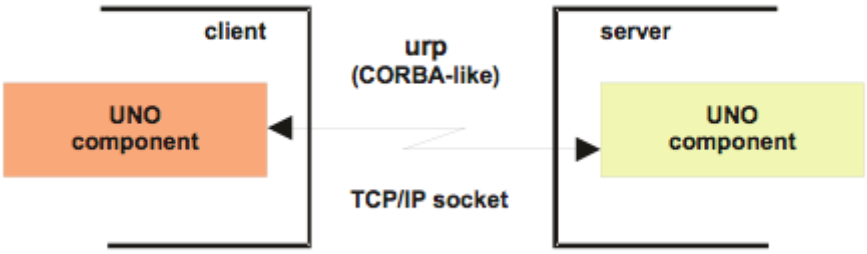

Figure 2: Client-/Server-Communications with UNO Components [FLAT05]

The term CORBA within the figure 2 means Common Object Request Broker Architecture. All applications of OpenOffice.org include specific components as well as common components that can be used by every application or foreign components from other applications. Specific components are for example the text document of the Writer application or the spreadsheet of the Calc application which are used esspecially by one specific program. In contrast to these exist common components such as printer settings. One feature which is part of every component is very amazing, its the provision of equally interfaces through which the system becomes very flexible and modifiable.
[OPEN05]

2.2.1.2 UNO Service Components

The architecture of OpenOffice.org consits basicly of service managers, a certain component context, objects, services, interfaces and properties. Generally every UNO component is represented by a specific service, which includes additional services, properties and interfaces.
[OPEN05]

2.2.1.3 Service Manager

The Service Manager is one of the most important parts of the UNO concept, it can be described as a “factory” which is producing services, illustrated in figure 3. Therefore it is the root component within the concept and has to be started before all the others. This component can be seen as an entry point for any application of UNO because it is responsible for the establishment of connections to UNO. The services created by the

Service Manager are objects of UNO that have specific abilities and tasks. Such a Service always exists in a component context which is composed of the Service Manager and the data used by the service. The central service manager of all UNO applications is the com.sun.star.lang.ServiceManager which is described by the following quote:
[OPEN05]

“The com.sun.star.lang.ServiceManager is the main factory in every UNO application. It instantiates services by their service name, to enumerate all implementations of a certain service, and to add or remove factories for a certain service at runtime. The service manager is passed to every UNO component during instantiation.”
[OPEN05 page 88]

Additional to the main “factory” there exist a main interface that is used by service managers, the com.sun.star.XMultiServiceFactory. This interface provides three methods, the createInstance() method that instantiates a default service, the createInstanceWithArguments() which creates a service instance consisting of supplementary parameters and the getAvailableServiceNames() method used to get all supported servicenames by the service manager.
[OPEN05]

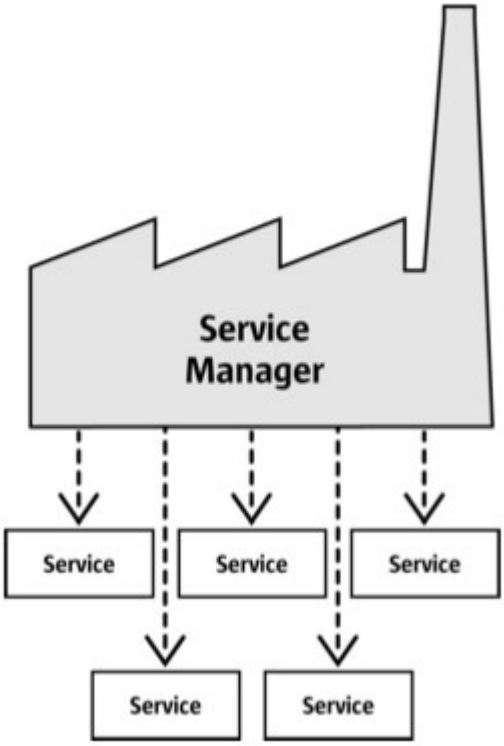

Figure 3: Service Manager [OPEN05]

2.2.1.4 Component Context

The service manager is the main factory as descripted before and is passed to every instantiated service. The weakness of this concept is that in many cases the deployed component has the need for more exchangeable functionality or information and the service manager is limited in this respect. For that reason the component context was introduced, which will become the main object in every application. Generally spoken the component context is a read-only container providing named values, as for example the service manager. Simply spoken it consists of the service manager as well as other data which is used by the services. This concept can be described as an environment in which components are living that have relationships, illustrated in figure 4. It is passed to a component during the process of instantiation and supports only the com.sun.star.uno.XComponentContext interface with the two methods getValueByName() and getServiceManager().
[OPEN05]

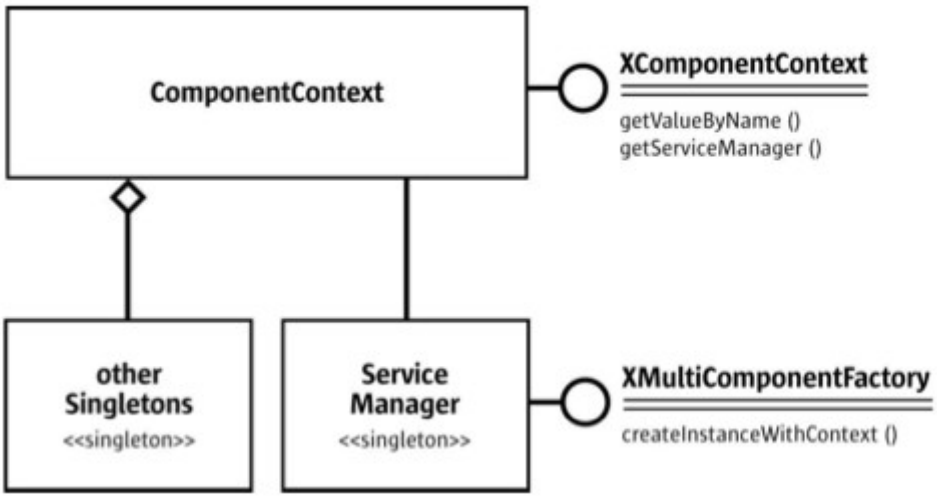

Figure 4: ComponentContext [OPEN05]

2.2.1.5 Objects

"In UNO, an object is a software artifact that has methods that you can call and attributes that you can get and set. Exactly what methods and attributes an object offers is specified by the set of interfaces it supports." [OPEN05 page 39]

Objects are very important for the whole concept because they are required to work with OOo, new objects are created by the service managers. They can for example represent an opened document or they can even hand out other objects. The specification, which gives information about the methods and attributes, of an object is defined by the set of interfaces it contains.
[OPEN05]

2.2.1.6 Services

Services provide a description of objects by an abstract object specification that consists of a set of interfaces and properties. In the context of Services the API of OpenOffice.org makes use of single- and multiple-inheritance. By reason of the fact that normaly objects are having many aspects, and a single-inheritanced interface provide only a description of one aspect, UNO makes use of multiple-inheritance services and interfaces to give an entire specification of objects. This is achieved by grouping all various aspects of an object in one multiple-inheritance interface type. Figure 5 gives an overview of how a language

dependent service makes use of the interfaces of an language independent service with a lot of services. In this case the language dependent service only needs to support one multiple-inheritance interface to inherit all of the other interfaces. This should give an idea how the relationship between services and interfaces is working in OOo.
[OPEN05]

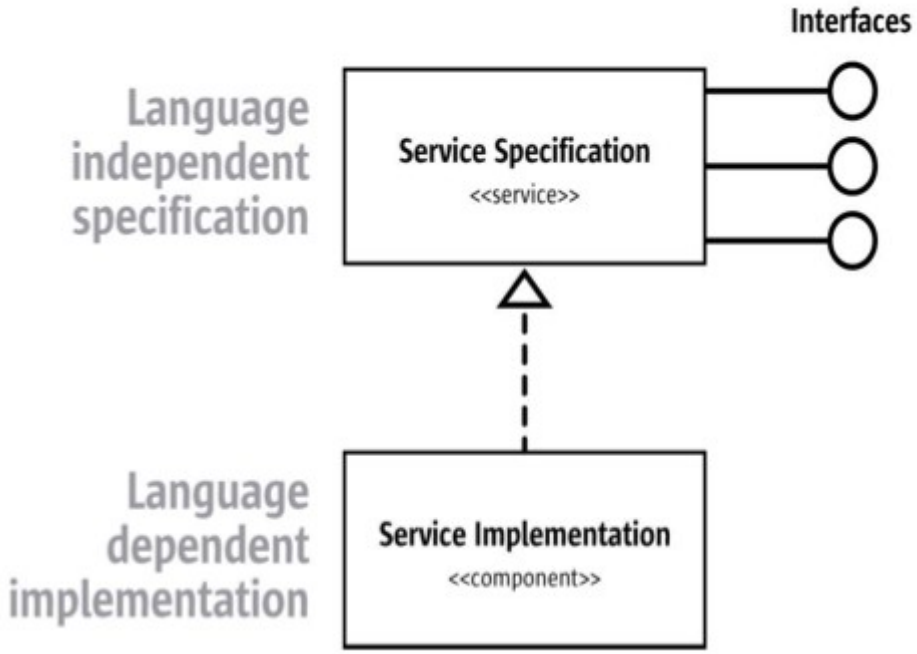

Figure 5: Inheritance of interfaces in OOo [OPEN05]

2.2.1.7 Interfaces

An Interface is one single part of an object and consists of a set of methods and attributes. Interfaces have the possibility to inherit one or more other interfaces, so that the reuse of an interface specification is possible. Since version 2.0 of OOo there is the possibility of multiple inheritences. Interface attributes are included in UNO to give extra support because of the widespread concept of setting and getting values. Furthermore interfaces give the designer more possibilities to express small differences between object features, for example features that are not integral parts of an object.
[OPEN05]

2.2.1.8 Properties

A property is a feature of a service, but it is not an integral or structural part of it. Thats the reason why the general methods getPropertyValue() and setPropertyValue() of the interface XPropertySet are used to manage properties. Every interface that supports the interface com.sun.star.beans.XPropertySet, which is the most basic interface for handling properties, can make use of the two methods. There are also other interfaces existing, like com.sun.star.beans.XMultiPropertySet, that also support the handling of properties.
[OPEN05]

2.2.1.9 Structs

A struct consists of several related properties and are similar to Java classes that are only including public member variables. In the API of OpenOffice.org they are used for the creation of compound UNO types. They are initialised using the message operator, for instance in Java the "." (dot) operator or in ooRexx the "~" operator.
[OPEN05]

2.3 Object REXX (Restructured extended executor)

2.3.1 History

Object REXX is an object orientated scripting language based on Smalltalk which was created by IBM. It was released in 1997 as the successor of Rexx, which has a syntax that is simple and easy to learn. The roots of Object REXX are situated in the 80'ies, in which companies who were using IBM mainframes and REXX started to use IBMs OS/2 operating system. In that time there was starting a huge object-oriented movement, so that IBM has to deliver a user interface capable of the new requirements. For this, engineers of IBM started to create an object orientated version of REXX. To fullfill this task they worked in cooperation with IBM user groups, especially with the so called "SHARE" special interest group (=SIG). The goal of the venture was to get a version of REXX with true object orientated features and the ability to be backwardly compatible to the predecessor REXX. In the year 1996 NetRexx was developed which has the same syntax as the original version of REXX but generates internally a Java code. The current version is

OpenObjectRexx which is an open source project of Rexx Language Association (RexxLA) , that offers a free utilisation of Object REXX.
[FLAT06-1]
[OORE06]

2.3.2 Overview

As mentioned before Object REXX is mainly based on Smalltalk and slightly on C++. It is completly compatible with the original version of Rexx and operates in an object orientated way. Internally commands are translated from procedural to object orientated. The object model is very powerful, it makes use of an interpreter and doesn't compile the code. The goal of Object REXX is to offer the user a powerful language which is easy to learn and use because of a human orientated syntax. ooRexx is available for a high number of operating systems, for instance Windows, Linux, MAC/OS or OS/2. The range of possibilities which is given through this scripting language can be best described by the following quote:
[OORE06]

"*ooRexx allows defining (meta-)classes, using reflection, creating one-off objects, mandating the use of explicit message operators for sending messages to objects, that look for methods by the name of the received message, as well as creating "floating" methods and employing a runtime environment that is realized as a stack of at least four directory objects being looked up on behalf of ooRexx programs, as well as being able to execute objects in a multithreaded manner*."
[FLAT06-1 page 5-6]

As the quote points out ooRexx offers single and mutiple inheritances, classes, objects and methods as well as messaging and polymorphism. It has included a set of very useful classes arranged in a flat classification tree, for instance the "Array", "Message", "Stem" or "String" class. The most important features of ooRexx are that it is a human orientated language based on english, so many of the provided instructions are meaningful like "SAY", "REQUIRES" or "EXIT". There are no strict rules about the formatting of the sourcecode, like in other languages, and it is not case-sensitive. Another important feature

is that it makes use of typeless variables, so there is no strict type definition. Furthermore it offers meaningful error messages and explanations when an error happens while interpreting the sourcecode.
[FLAT06-1]

2.3.3 Syntax Overview

This paragraph provides a short introduction to the syntax of Object REXX, it should give the user a better understanding to be capable of reading the sourcecode of the work. There are two possible ways to write comments, the first is used to make a comment just in one line, for example "*-- word"*. The second is used if an annotation takes more than one line, at the start and the end there must be set a special character like */* word */*. As mentioned above ooRexx has no strict typing, this means that variables are defined only by assigning a value of any type to it, for instance "*a = 23"*. The keyword "SAY" is responsible for printing a value of any kind in the command line, like "*SAY a"* or "*SAY Hello".* A block in Object REXX is an instruction which includes a not defined number of instructions, it starts with the keyword "*DO"* and ends with the keyword "*END"*. The IF instruction is realised similary to other programming languages, *IF condition THEN instruction ELSE instruction.* Loops are also realised in a very easy and common way like this example:

```
i=0
DO WHILE i < 10
        instruction
        i = i + 2
END
```

Also the creation of arrays is very easy to fullfill with the ooRexx syntax:
Creation of an new array: *array = .array~new*
Inserting of values into the array: *array[1] = value*

Different to other programming languages Object REXX is using the "~" character as the message operator, for example the creating of an object would look like this:

object~new("name")

There also exists the possibility to use the message creator to create cascaded messages by inserting the "~" character two times. With the help of cascading it is possible to call several methods at the same time like displayed beneath.

object~~method_1()~~method_2()

On the left side of the message operator stands the object where the message is sent to and the right side represents the message which is transmitted to the object. If the message is for example a method which includes arguments, then after the message name round parenthesis are added that contain these arguments. When the message is sent and the object has received it, it is looking for a matching method starting from the class of the instantiated object searching every superclass until finally the root class, the so called "Object" is reached. The searching is finished when the first found method is called by the object or when the method can't be found.

Procedures can be created by using directives, like using the keyword *::ROUTINE* or by writing the name of the routine followed by one colon, for example procedure:. The keyword *CALL* is needed to invoke procedures and also for the execution of built-in functions.

Directives are very important components of ooRexx, they are defined by two colons "::" and used for classes and their methods. It instructs the ooRexx interpreter to start them, before the rest of the program is carried out. This is a very good way to assure that before a program is executed all required resources are made available by the interpreter. A very good example in the context of OpenOffice.org is the UNO interface which is implemented by ::REQUIRES UNO.cls. This class file is a helper class of ooRexx that is responsible for the automation of common steps. For instance it has the ability to sum up a set of method calls to get a connection to OpenOffice.org and the component context service. At the time of creation a method called "init", which has the purpose to operate as a constructor, is

invoked. This means that every message which is sent to the “new” method is forwarded to the constructor.

2.4 BSF4Rexx

BSF4Rexx is the “Bean Scripting Framework” for Object REXX, it was made at the university of Essen by Prof. Flatscher and the student Peter Kalender. It is based on the “Bean Scripting Framework” which is an open source project of IBM. BSF gives scripting languages the possibility to operate within Java applications and to use on Java components like objects and functions. It consists mainly of the BSFManager and the BSFEngine.

- The “BSFManager” deals with all scripting execution engines that are running under its control. It sustains the object registry which provides scripts access to Java objects. With the use of this component Java applications are able to get access to scripting services by creating an instance of the “BSFManager”.

- The “BSFEngine” offers an interface which has to be implemented for a language that make use of BSF. The provided interfaces by “BSF4Rexx” are used to get a connection between Java and scripting languages. The engine uses a common interface for all supported scripting languages.

Since its introduction there were three versions of BSF4Rexx, the first one was the “Essener version” which gave Java developers the possibility to use Object REXX as a scripting language. The second version called “Augsburger version” from 2003 provides programmers with the advantage of handling Java classes from Object REXX. The newest version is the “Wiener version” which doesn't request strict typing and offers a lot of new functions to automate OpenOffice.org. Figure 6 shows the architecture of the “Vienna Version”, esspecially the communication between ooRexx and Java using the BSF.cls.

[FLAT06-2]

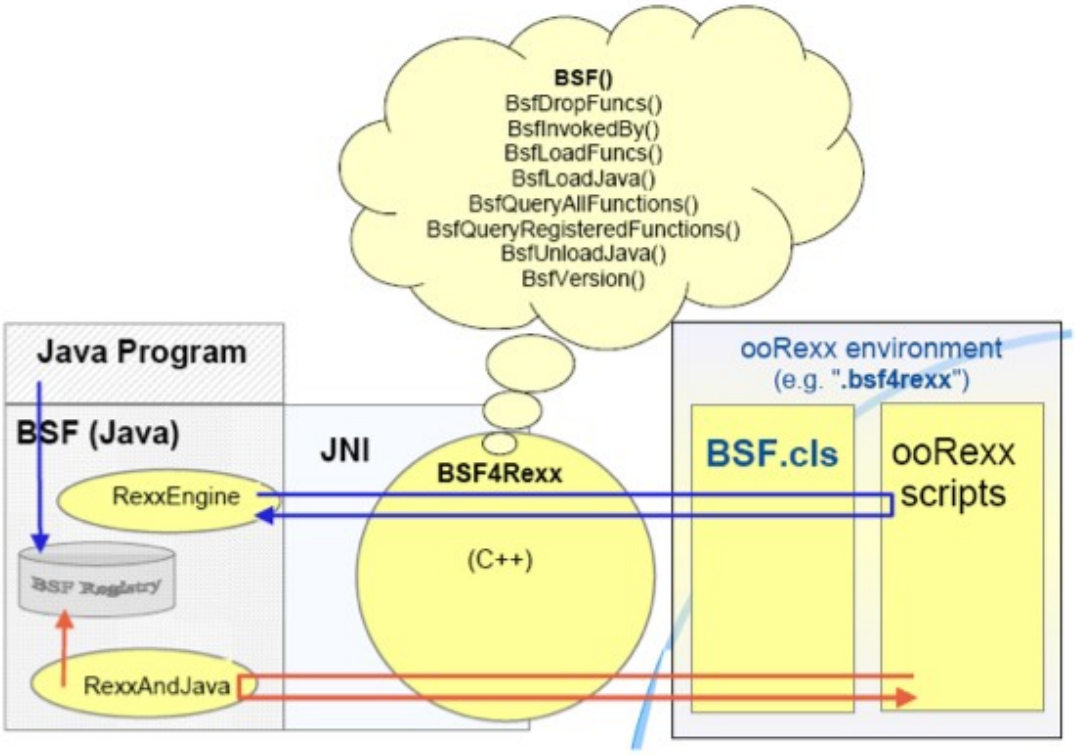

Figure 6: Architecture of BSF4Rexx [FLAT06-2]

Two important modules of BSF4Rexx should be mentioned, one is the UNO.cls and the other one the BSF.cls.

The UNO.cls class supports the UNO component model. It is responsible for the communication with OpenOffice.org and it makes the automation of common steps with the help of the BSF.cls much easier.

The BSF.cls file provides the bridge to access Java for example within a scripting language like ooRexx. The public class BSF is the ooRexx proxy class responsible for the representation of Java classes. A high number of the BSF4Rexx subfunctions can be called through the public class BSF as class or instance methods, like bsf.createArray(). The initialitsation of the UNO.cls or the BSF.cls is achieved through using the “CALL” or the “::REQUIRES” statement. The newest version of BSF4Rexx provide a lot of new functions that make it possible to reduce the amount of code by the factor 3. Figure 7

shows a code example without using BSF.cls and figure 8 an example which uses BSF.cls. The comparison of this two code samples gives an idea how big the amount of code reduction is.

[FLAT06-2]

```
 /* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */

     /* load the BSF4Rexx functions and start a JVM, if necessary  */
if rxFuncQuery("BSF") = 1 then   /* BSF() support not loaded yet ? */
do
  call rxFuncAdd "BsfLoadFuncs", "BSF4Rexx", "BsfLoadFuncs"
  call BsfLoadFuncs /* registers all remaining BSF functions       */
  call BsfLoadJava  /* loads Java                                  */
end

say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')
```

Figure 7: Interfacing Java without BSF.cls [FLAT06-2]

```
/* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */

say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')
::requires bsf.cls  /* load the Java support */
```

Figure 8: Interfacing Java with BSF.cls [FLAT06-2]

2.5 Text Documents

Due to the fact that the programming part deals exclusively with the Writer application and therefore with text documents this chapter gives an overview of the architecture of such documents in OpenOffice.org. Furthermore it will give a description of special components of the Writer application that have been used inside the program and that are important to be mentioned from my point of view. This should give a better understanding of the sourcecode explanations.

2.5.1 Overview

Developers of OpenOffice.org are working directly with the text document model, which has a controller object that allows to manipulate the visual presentation of the text document.

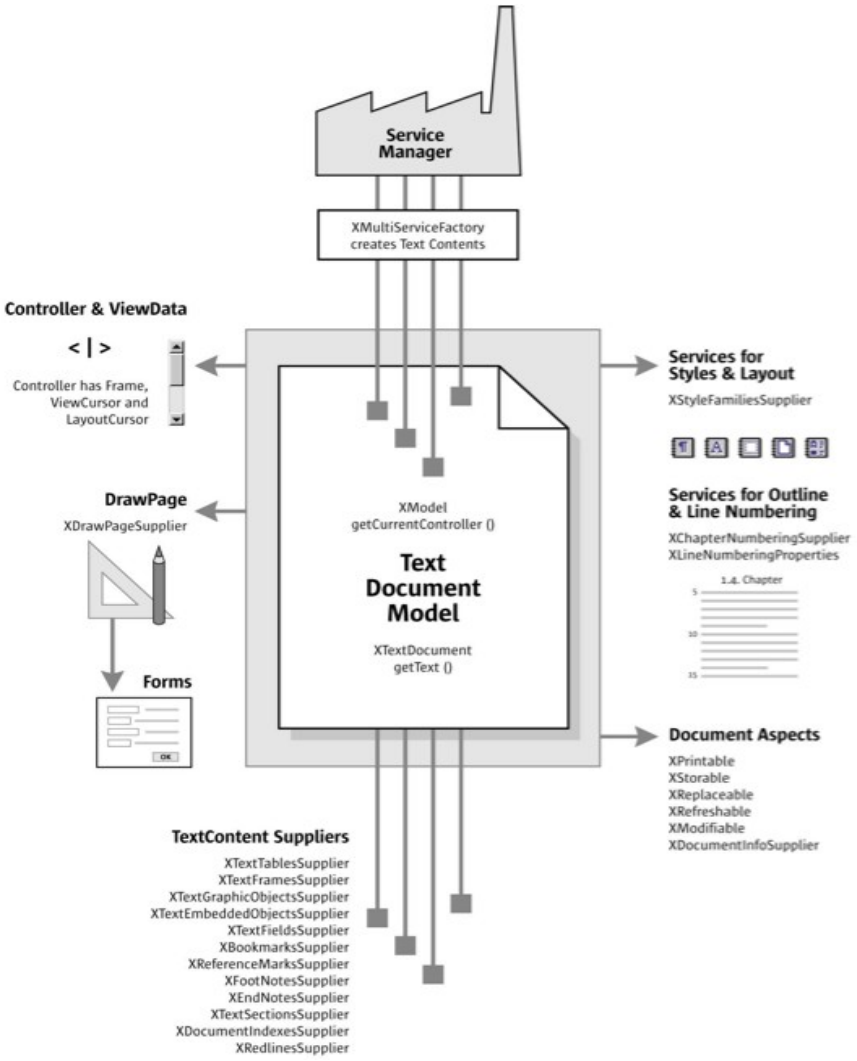

Figure 9: Text Document Model [OPEN05]

The controller serves two purposes, the first is to move through the text document via a visible text cursor or changing the zoom. The second is to get information about the current view status, like the current selection, page, total line count or page count. The model generaly consists of five major areas, illustrated in figure 9.

The five architectural parts are “text”, “service manager”, “draw page”, “text content

suppliers” and “objects for styling and numbering”.

- Text
  The main element of the Text Document Model is the Text, that consists of character strings that are arranged in paragraphs and other possible text contents.

- Service Manager
  The Service Manager is responsible for the creation of all text contents that are not included into the Text area, for instance text tables, text frames or graphic objects. The architecture of OpenOffice.org is designed to offer each document model its own Service Manager, thats why this manager is different to the main Service Manager. It is very important not to confound this two different forms of Service Managers.

- Text Content
  The text contents which are created by the Service Manager can be recalled by the Text Content Supplier.

- DrawPage
  The DrawPage lies above the text because it is a transparent layer with contents that have the ability to change the underlying text. For example it has the ability to force the text to wrap around parts of the DrawPage. The DrawPage is also able to retrieve content of its layer which can't be done by the Text Content Supplier.

- Objects for Styling and Numbering
  The last point are objects for styling and numbering, they are services used for styling and structuring the text content.

[OPEN05]

An UML chart of the OpenOffice.org API that gives an overview of the com.sun.star.text.TextDocument service and its interfaces is shown in figure 10. Additional the relationship to the com.sun.star.document.OfficeDocument service is illustrated which offers interfaces that every TextDocument must include. The whole chart shows the basic obligatory elements of a TextDocument. This means that the TextDocument includes text, it consits of a model with an URL and a controller, is searchable, refreshable, modifiable, printable and storable.

[OPEN05]

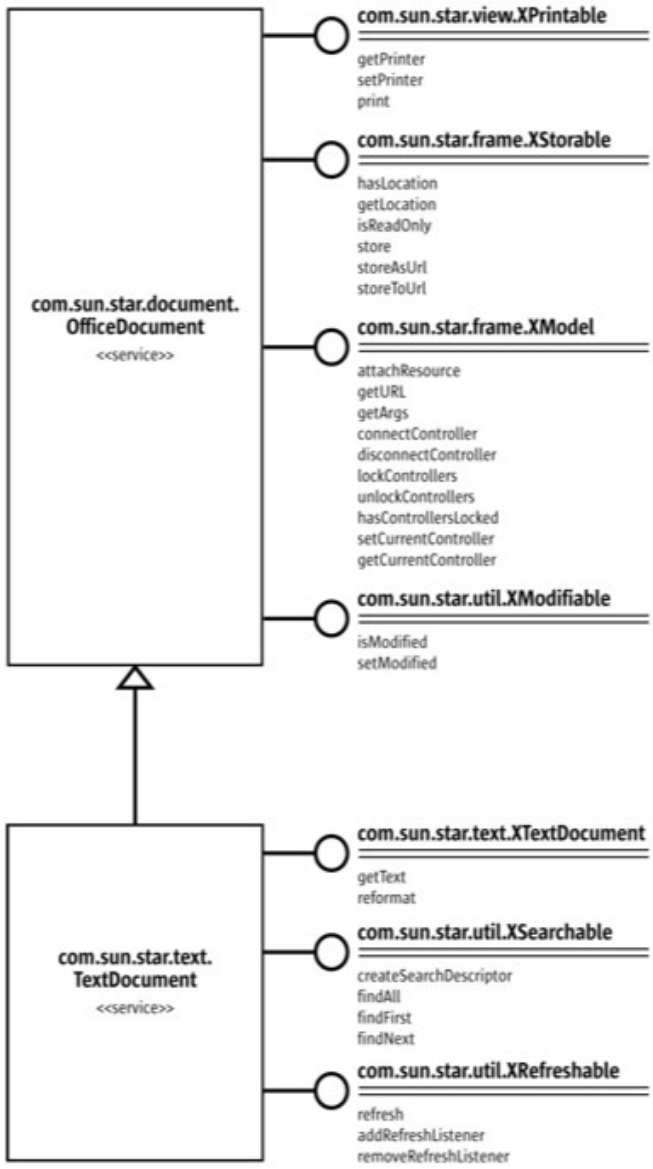

Figure 10: TextDocument interfaces [OPEN05]

2.5.2 Handling Text Content

The com.sun.star.text.XText is the primary text content interface, this means that every text content is implemented with the help of the XText interface by using its methods. "*The primary purpose of a text object is to contain text content, create text cursors to move through text, insert text, and remove text.*"
[PITO04 page 286]

Table 1 gives an overview of the set of methods offered by this interface, it has to be pointed out that the first four method are offered through the XSimpleText interface and only are available because their interface is extended by the Xtext interface.

Method	Description
createTextCursor()	Return a TextCursor service used to traverse the text object.
createTextCursorByRange(XTextRange)	Return a TextCursor that is located at the specified TextRange.
insertString(XTextRange, String, boolean)	Insert a string of characters into the text at the specified text range.
insertControlCharacter(XTextRange, Short, boolean)	Insert a control character, such as a paragraph break or a hard space, into the text.
insertTextContent(XTextRange, XTextContent, boolean)	Insert text content sucha as a text table, text frame, or text field.
removeTextContent(XTextContent)	Remove the specified text content from the text object.

Table 1: Methods offered by the XText interface [PITO04 page 286]

Another interface which is used to describe the position of objects within the text is also very important for handling the content within the text. This interface is called com.sun.star.text.XTextRange and with its help the position of objects are specified through a text range that has a certain beginning and end. It is very important for the Xtext interface to be able to insert or retrieve text content from a specified position of the text. The included methods are displayed in table 2:

Method	Description
getText()	Return the Xtext interface that contains the text range.
getStart()	A text range has a start and end position. The getStart() method returns a text range that contains only the start position of this text range.
getEnd()	A text range has a start and end position. The getEnd() method returns a text range that contains only the end position of this text range.
setString(String)	The setString() method replaces all of the text between the start and end positions with the argumet string.
getString()	Return a string that represents the text in this text range.

Table 2: Methods offered by the XTextRange interface [PITO04 page 286]

2.5.3 Cursor

Cursors of a text document are used to access the content of the text, in OpenOffice.org there exist two kind of cursors the ViewCursors and the TextCursors.

2.5.3.1 ViewCursor

These kind of cursor, as the name implies, are dealing with visible cursors within a text document. In the work only the XtextViewCursor is used to retrieve information about the position of text contents, to give a better overview all available forms are mentioned.

- XviewCursor – com.sun.star.view.XViewCursor
  This interface provides the simplest view cursor which is only able to move up, down, left and right within the text as well as tables.

- XTextViewCursor – com.sun.star.text.XTextViewCursor
  The XtextViewCursor offers a cursor that is positioned in a view of a text document. It is able to be set visible or hidden and retrieve its current position through the set of methods that is offered.

- XLineCursor – com.sun.star.view.XLineCursor
  The interface enables the visible cursor to move line by line throught the text.

- XPageCursor – com.sun.star.text.XPageCursor
  The set of provided methods gives the cursor the opportunity to move between pages.

- XScreenCursor – com.sun.star.view.XScreenCursor
  This interface enables the cursor to move through the document by visible pages.

[PITO04]

2.5.3.2 TextCursor

The TextCursor is a main element which is used very often during the whole work for accessing the content of the documents that are dealed with. It must be mentioned that in contrast to the view cursors these type of cursors are only used to access the data but has no information about its presentation. The service includes the interface XTextrange, responsible for the accessing of the string content. Additionally there are many exported interfaces, the most important in our case are the XTextCursor, XWordCursor, XsentenceCursor and the XParagraphCursor interface.

- XTextCursor - com.sun.star.text.XTextCursor
  The interface describes the position of an object within the text with the help of the XTextRange interface that specifies the position. Certain methods give the possibility to move the Cursor character by character through the text.

- XWordCursor - com.sun.star.text.XWordCursor
  The XwordCursor is similary to the XtextCursor interface, the difference is that the methods given through this interface let the cursor move word by word.

- XSentenceCursor - com.sun.star.text.XSentenceCursor
  This cursor interface is also equally to the two interfaces before in some respect, with it the cursor is able to move from sentence to sentence with the help of several methods.

- XParagraphCursor - com.sun.star.text.XParagraphCursor
  The last cursor interface is used like the name indicates to move through paragraphs and provide methods which can be used to fulfill this task.

[PITO04]
[OOAP06-1]

2.5.4 Loading documents

The loading of documents is achieved with the interface com.sun.star.frame.XComponentLoader which is implemented by the Desktop service of OpenOffice.org. This interface includes only one method the loadComponentFromURL() which is used to load documents from a specified URL. When this method is called it gets a sequence of com.sun.star.beans.PropertyValue. This sequence is send by a parameter who is implementing the service com.sun.star.document.MediaDescriptor. The MediaDescriptor is very important for the loading of documents because it specifies the information from where a resource should be loaded and includes properties that are able to specify the way documents should be loaded or saved. The usage of the loadComponentURL() is returning a com.sun.star.lang.XComponent interface. For the loading of empty documents there exists the parameter URL that is a parameter for the loadComponentFromURL() which overrides all properties passed to the MediaDescriptor. Figure 11 gives an overview of all available URL parameters.

Component	URL
Writer	private:factory/swriter
Calc	private:factory/scalc
Draw	private:factory/sdraw
Impress	private:factory/simpress
Database	.component:DB/QueryDesign .component:DB/TableDesign .component:DB/RelationDesign .component:DB/DataSourceBrowser .component:DB/FormGridView

Figure 11: URL parameters [OPEN05]

2.5.5 Closing documents

As described before after loading a document there is the com.sun.star.lang.XComponent interface available which is used to control UNO objects. This interface includes a method called disposal() which is used to close the document. There exist also other methods and even interfaces that are dealing with closing operation, for example the com.sun.star.util.XCloseable interface which is used for frames and models.

[OPEN05]

2.5.6 Saving documents

After the loading of documents they are accessed using special interfaces typically for the used application, in our case it would be the com.sun.star.text.XTextDocument interface. This interface like all other office components is supporting the com.sun.star.frame.XStorable interface. The Xstorable interface includes the store(), storeAsURL() and storeToURL() method for saving documents. The store() method just stores the file to the location where it was loaded from, the storeAsURL() saves the file to the specified URL and makes the URL to the new location of the object. The storeToURL() stores the file to an URL but doesn't change the location of the object. The last two methods makes use of the MediaDescriptor for setting storing specifications.

[OPEN05]

2.5.7 Printing documents

The printing functionality is a very common functionality and is supported by every component within OpenOffice.org. For that reason text documents support the com.sun.star.view.XPrintable interface that provides methods to modify the printer settings and is able to start printing jobs. The methods for modifiying the printer is the getPrinter() and the setPrinter() method. The printer is set with the help of the properties of the com.sun.star.view.PrinterDescriptor interface. The print job is started by the method print().

[OPEN05]

2.5.8 Search&Replace of text content

The com.sun.star.util.XSearchable interface is supported by the writer model and is used for searching the content with provided methods. First of all a SearchDescriptor is created to be able to search the document, after that it is possible to set the searchstring with methods of the XSearchDescriptor interface. Next the com.sun.star.util.XSearchDescriptor is used in combination with the Xsearchable interface and enables through the two methods getSearchString() and setSearchString the getting and setting of the string which has to be searched for. Finally the search process is realised by the one of the find methods, findAll(), findFirst() or findNext().

The interface com.sun.star.util.XReplaceable is used for the replacing of certain strings included in the text. It includes the methods createReplaceDescriptor() and replaceAll() and its functionality is similary to the one of the Xsearchable. The interface provides the possibility to search after a string and replace it with a specified string. Therefore it can use the methods of the XSearchDescriptor to set for example a SearchString. Similary to the XSearchDescriptor interface the com.sun.star.util.XReplaceDescriptor provides methods to set and get the ReplaceDescriptor, which are called getReplaceString() and setReplaceString(). After setting of the search and replace descriptor the process is started by calling the replaceAll() method.

[OPEN05]

2.5.9 Control of Shapes

A main part of the program deals with the insertion of forms in the text document, to be able to do this the control of the shape which is going to be inserted is needed. This is achieved by using the services Shape and Controlshape. ControlShape services are tied to control models and they are able to insert form control models to a document. The service com.sun.star.drawing.ControlShape includes the service Shape which specifies all characteristics of Shapes. The exported interface XControlShape is needed to get access to the controls model. It includes the method getControl() which returns the control model of the current shape, as well as the setControl() method that is setting the control model for a shape. The XControlShape interface can also make use of the methods of the XShape interface and therefore set the characteristics of the shape like size or position with the methods setsize() and setposition() of the XShape interface for instance.

[OPEN05]

2.5.10 Dispatch Process

This Dispatch framework is in generaly used for the communication between an office component and an user interface by handling command executions and the provision of attribute information from office components. All interactions of user are commands which are executable and can be called within the framework using the command URL from the struct com.sun.star.util.URL. Such URLs are string values that are following a certain scheme, like *file*: or *http*:. In our case the command for switching a button of the control model of a form shape is assigned to a URL. The URL is handled by the XDispatchProvider, before this can be done the URL has to be parsed with the help of the com.sun.star.util.URLTransformer service. After this task the URL is syntactly complete and ready to be executed. The method queryDispatch() of the XDispatchProvider helps to get the dispatch object for commands by looking for executable commands under the specified URL. The interface also implements com.sun.star.frame.XDispatch which includes the dispatch() method that is finally used to dispatch the certain URL and in our case is switching the button.

[OPEN05]

2.6 Software Requirements

The macro was built to be executeable under Windows and Linux. Concerning the software requirements it needs a Java, OpenOffice.org, Object REXX and BSF4Rexx to be installed on the system. The macro was tested under the current version 6 of Java, version 2.1 of OpenOffice.org, version 3.1.1 of ooRexx and the “Wiener” version of BSF4Rexx. It was also tested with some of the former versions and should be backwardly compatible. All of the mentioned applications are available for Windows and Linux. One important thing to do when installing is to make sure that the classpath and the pathes are set corecctly.

2.6.1 Java

The current version of Java is at the moment in december 2006 version 6.0 and can be downloaded for free on the official webpage http://java.sun.com. Normaly it isn't needed to set a classpath or a path to the directory but if there are any errors occuring concerning Java it is recommended to set them. Insert the classpath for the main directory of Java and the path to the Java Virtual Machine.

2.6.2 OpenOffice.org

The office suite is downloadable for free on the offical website http://www.openoffice.org, the current version in december 2006 which was used for the automation is OpenOffice.org 2.1. One important thing which has to be done after the installation before it is possible to automate OpenOffice.org via ooRexx is to install the OpenOffice.org support of BSF4Rexx and to set the required classpathes to the required “JAR” files of OpenOffice.org. The classpath can be set manual or by executing the setEnvironment4OOo of BSF4Rexx and then copying the output into the classpath of the system.

2.6.3 Object Rexx

Since Object Rexx has become open source it is freely available on its webpage http://www.oorexx.org. The current version is 3.1.1 and there should be no need of setting classpathes like in Java. However if there are any problems set classpathes for the main directory of Object Rexx and the OODialog directory within the main directory.

2.6.4 BSF4Rexx

Because of the reason that the versions of BSF4Rexx are changing frequently, it is advised to be very carefully when choosing the right version. The current version can be downloaded under the following url http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/. There you get a directory with different files including the installation files, the sourcecode and additional reading material. Download the BSF4Rexx_Install.zip file unzip the archiv and follow the instructions in the two textfiles readmeBSF4Rexx.txt and readmeOOo.txt very carefully to install BSF4Rexx correctly.

When all of the above software installations are completed and the test scripts of BSF4Rexx are executeable without any error the system is ready to execute the macro script.

3 Realisation

The target of the project was to deliver a macro script that is able to handle templates. Due to the reason that all used technonlogies for the solution are available under windows and under linux, the script is able to work on both systems, only the file pathes within the script have to be changed.

3.1 Overview

The developed solution works as follows: The user gets a graphical interface using Java Swing to control the macro. This is needed because at the beginning of the macro the user is able to choose between creating a new document with the latest template or to load a historical one.

The first case is activated when the user chooses the creation of a new document. In this case the script searches for the latest template within a specified directory, opens the template, searches for special keywords that include the information at which position to set the textboxes. After that a textsection with content protection is inserted to protect the document against any changes of the user excepted the content of the textboxes. After the creation of the new document the macro stops and creates another graphic interface that informs the user that the macro is in holding position. The user is now able to insert the content into the document. After the finalisation of the input the user can reactivate the macro by pressing a button which is provided by the graphic interface. The macro saves the name of the template and the inserted content of the textboxes to a new Writer file, than it sets the borders of the textboxes invisible, exports the document to PDF and print it to the standard printer of the workstation. After all of these steps the creation of the new document is finished and the user gets a notification.

The second case that can be chosen is used for loading historical documents and to merge them, so that the user is able to use them. When the user is pressing the historical button a new graphic interface is opened which gives an overview of all existing documents. The user has to choose the needed document to open it by pressing the button beside the name of the document. The script opens the content file and searches

for the name of the template and the content. The final step is to open the template, to insert the textboxes and the content so that it is possible to work with the document.

The program starts as mentioned before with a Java Swing JWindow that gives the user the option to choose between two possibilities which were described before. The rest of the script consists of routines which are called within the program to get a better control of the work flow and to get a structur into the sourcecode. The used routines are:

- Routine main_new: It is responsible for the opening of the current template, for the preparing of the document, as well as for the saving, exporting and printing.

- Routine main_old: This routine is used for the retrieving of a historical document, esspecially for the assembling of the content with the right template.

- Routine state_window: The state window is used for turning the macro into holding position, so that the user has time to insert the content to the document. The JWindow also includes a continue button which is used to finish the program and for closing the JWindow.

- Routine switch_design_mode: It is needed for the switching of the design mode of the form control from design to live mode, so that the form can't be changed any longer.

- Routine textsection: The textsection as the name indicates creates a protected textsection across the whole document to protect it againt changes from the user.

- Routine create_form: This routine is used for creating and intializing a shape for the document and to be able to insert textfiels.

3.2 Sourcecode Documentation

This part of the work gives a detailed documentation of the sourcecode as well as explanations and the provision of background information.

3.2.1 Initialisation

The programm makes use of many Java classes to provide the user a graphical interface to control the macro, which is shown in figure 12.

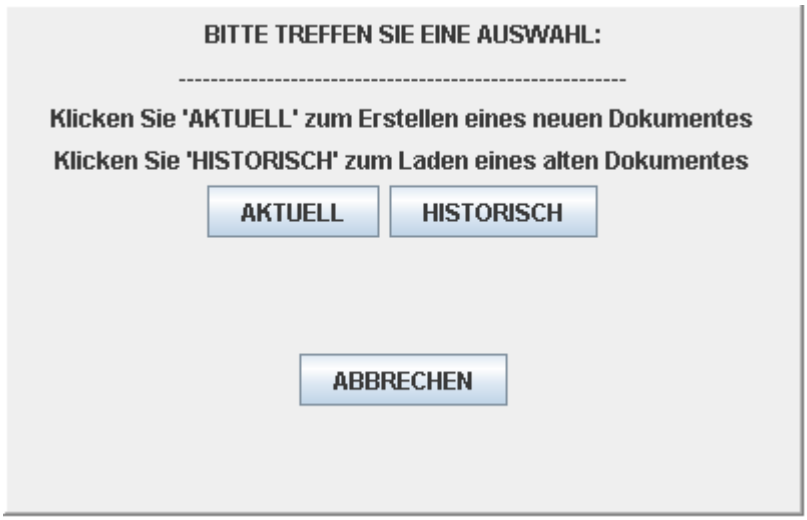

Figure 12: Main menu template handling tool

Therefore at the beginning of the program all of the needed Java classes are imported with the help of the bsf.import() class method of BSF, illustrated below in figure 13. In the first part of the script the create_gui routine is called and all of the needed classes are loaded to be available at the beginning of the script, with the help of the **::Requires** keyword. The script makes use of the BSF.cls , the UNO.cls and the RXREGEXP.cls support. The rest of the script is handeled with routines that are called within the main part starting out from the first JWindow.

```
-- Import of all needed Java classes
.bsf~bsf.import("java.awt.GridLayout","GridLayout")
.bsf~bsf.import("java.awt.GridBagLayout","GridBagLayout")
.bsf~bsf.import("java.awt.Toolkit","Toolkit")
.bsf~bsf.import("java.awt.ScrollPane","ScrollPane")
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout")
.bsf~bsf.import("java.awt.GraphicsEnvironment", "GraphicsEnvironment")
.bsf~bsf.import("javax.swing.JFrame","JFrame")
.bsf~bsf.import("javax.swing.JLabel","JLabel")
.bsf~bsf.import("javax.swing.JRadioButton","JRadio")
.bsf~bsf.import("javax.swing.ImageIcon","Icon")
.bsf~bsf.import("javax.swing.JWindow","JWindow")
.bsf~bsf.import("javax.swing.JPanel","JPanel")
.bsf~bsf.import("javax.swing.JButton","JButton")
.bsf~bsf.import("javax.swing.JOptionPane", "JOptionPane")
.bsf~bsf.import("javax.swing.border.BevelBorder", "BevelBorder")
.bsf~bsf.import("javax.swing.border.EtchedBorder", "EtchedBorder")

Call create_gui

::Requires BSF.CLS        -- Make oo-like BSF4Rexx support available
::Requires UNO.CLS        -- Get UNO support
::Requires RXREGEXP.CLS   -- Get the support of regular expressions
::Routine makeUrl         -- operating system independent
Return  ConvertToURL(stream(arg(1), "c", "query exists"))
Exit
```

Figure 13: Code Snippet - Java classes import

3.2.2 Routine create_gui

The first routine of the script deals with the creation of a Java swing JWindow which allows the user to choose between the creation of a new document and the loading of a historical document. The JWindow is filled with JPanels that include JLabels with text and JRadiobuttons. The user can make a decision by clicking one of the JButtons.

The code part in figure 14 shows the creation of the Java objects and the setting of the Layout of the JLabels, all accomplished using Java methods. The Java objects are created with the function new(), the layout is set with the setLayout() and the objects are added with the add() method.

```
::Routine create_gui
-- Create a java swing menu with radiobuttons
-- Create java swing objects
label   = .JLabel~new("BITTE TREFFEN SIE EINE AUSWAHL:");
label2  = .JLabel~new("-------------------------------------------------------")

label3  = .JLabel~new("Klicken Sie 'AKTUELL' zum Erstellen eines neuen Dokumentes");
label4  = .JLabel~new("Klicken Sie 'HISTORISCH' zum Laden eines alten Dokumentes");
window  = .JWindow~new();
labelp  = .JPanel~new();
buttonp1  = .JPanel~new();
jbutton1 = .JButton~new("AKTUELL");
jbutton2 = .JButton~new("HISTORISCH");
jbutton3 = .JButton~new("ABBRECHEN");

buttonp2 = .JPanel~new();
bevelborder = .BevelBorder~new(0)

-- Set the Layout and add objects to jpanels
labelp~setLayout(.FlowLayout~new())
labelp~~add(label)~~add(label2)~~add(label3)~~add(label4)
buttonp1~setLayout(.FlowLayout~new())
buttonp1~~add(jbutton1)~~add(jbutton2)
buttonp2~setLayout(.FlowLayout~new())
buttonp2~~add(jbutton3)
```

Figure 14: Code Snippet - Routine create_gui

The next task is to set the location of the JWindow according to different screen resolutions that the graphic interface adapts automatically to different screen settings, shown in figure 15. This is accomplished with the use of the Java Toolkit class that includes the method getScreenSize() which returns a dimension object that includes the current screen resolution in pixels. The access to the default toolkit of the superclass is achieved through the method getDefaultToolkit(). The width and height values of the object are retrieved calling width() and height(). They are used to calculate the exact centered position of the JWindow on the screen. The size of the JWindow is also fixed to a minimum level and the possibility to rise if a higher screen resolution is used. This is accomblished by fixing the variable size to the screen resolution.

The contentPane for the window is retrieved by the method getContentPane() of the jwindow class, this step allows now the adding of JLapels and the setting of the Layout. Furthermore the location, size and visibility is set with the methods setLocation(), setSize() and setVisible(). The window should be always on top this is achieved by using the method setAlwaysOnTop().

```
-- Set the position of the jwindow to the center of the screen
toolkit=.Toolkit~getDefaultToolkit()
screenSize = Toolkit~getScreenSize()

wx = screenSize~width()/3.2
wy = screenSize~height()/4
Parse var wx wx '.'
Parse var wy wy '.'

If wx<400 then wx=400
If wy<256 then wy=256

w_x = wx/2
w_y = wy/2
Parse var w_x w_x'.'
Parse var w_y w_y'.'
scx = screenSize~width()/2-w_x
scy = screenSize~height()/2-w_y

-- Set the layout of the jwindow and add the jpanels
window~getContentPane()~setLayout(.GridLayout~new(3,1));
window~getContentPane()~setBorder(bevelborder);
window~getContentPane()~~add(labelp)~~add(buttonp1)~~add(buttonp2);

-- Set the location, size and visibility of the jwindow
window~~setLocation(scx,scy)~~setSize(wx,wy)~~setVisible(.true)-
~~setAlwaysOnTop(.true)
```

Figure 15: Code Snippet - Set JWindow position

After the creation of the JWindow two eventlistener are added, using the bsf.addEventListener() instance method of BSF, to the JButtons to execute actions by clicking them. A neverending loop is looking for any messages from the eventlistener and executes them as a rexx programm. The actions are advised to call procedures responsible for the further progression of the script, which can be seen in figure 16.

```
-- Add eventlisteners to the jbuttons
jbutton1~bsf.addEventListener("action", "", "Call setnew")
jbutton2~bsf.addEventListener("action", "", "Call setold")
jbutton3~bsf.addEventListener("action", "", "Exit")


/* A never ending loop which execute the messages from the
   eventhandler as a rexx program*/
Do forever
        event = bsf("pollEventText")
        interpret event
End

Exit
```

Figure 16: Code Snippet - Eventlistener

The first procedure is called setnew and is responsible for the retrieving of the latest template and the creation of a new document based on it. The code to this procedure is shown in figure 17. To get the latest template the SysFileTree function of the REXX

Utilities (RexxUtil) Dynamic Link Library package of ooRexx is used to get an array of the system file tree of a specified destination. The content of the array is parsed to extract the date information of the template which is included in each filename.

If the date is included at the beginning of the filename the last position of the array includes the latest template because the function SysFileTree is reading out the filenames as they are ordered in the file system. But if thats not the case a “Do” loop gets the actual date by the comparison of all available objects inside of the array and saving the array position of the latest template to the variable f.

The filename of the current template which is retrieved of the array is saved to the local environment, it would look like this example “.local~filename = value”. Through this it becomes a constant value which is available in the whole program like a global variable. It can be retrieved by the environment symbol which starts with a point followed by the name of the variable, in our case it would look like this: “.filename”. After that the main routine for the creation of a new document is called by the ooRexx **Call** keyword.

```
-- Procedure which gets the latest template and opens it
-- It is called from the eventhandler by pressing the jbutton "newb"
setnew:
        window~dispose()  -- Close the open jwindow
        CALL SysFileTree "C:\_templates\*.*", "file", "O"
        -- Creation of a stem(="array") of the system file tree of the specified folder

        Do i=1 TO file.0
                PARSE VAR file.i prefix.i =15 filename.i =23 subfix.i
                -- Extraction of the date information out of the full path of the file
        End

        i=2
        e=1
        f=0


-- Comparison of the Dates to find out the latest one
        Do Until i > file.0
                If filename.e>filename.i Then e=e-1;
                If filename.e>filename.i Then f=e;
                Else f=i;
                e=e+1
                i=i+1
        End

.local~filename =  filename.f""subfix.f
        Call main_new
Exit
```

Figure 17: Code Snippet - Procedure setnew

The second procedure which is activated by pressing the historic JRadiobutton is used for loading a historical document. Before the document is loaded another JWindow is created that also uses the SysFileTree function to save all historical documents within a stem, shown in figure 18.

Figure 18: Historical document menu

The script generates a JRadiobutton for each document which is inserted into a ScrollPanel with the help of a loop shown in figure 19. The JWindow is designed the same way as the first JWindow except the use of a ScrollPanel which is needed because the number of historical documents is unknown and therefore the size of the panel can't be fixed. Every JRadiobutton of a historical document includes, like before, an eventlistener but in this case there are two of them. One is used to set a variable to tell the eventlistener that the button was pressed and the other one saves the number of the jbutton within the loop to a variable so that it is possible to load the right document.

At the end a JPanel with Text and the ScrollPanel is added to the JWindow and the window is initialised the same way as the first one before. An old document is loaded by activating one JRadioButton and than pressing the “OK” button, which is executed by the use of the open procedure. The second JButton called “ABBRECHEN” is used to call the create_gui routine and simultaneously closing the current JWindow, so that the user can move back to the main menu.

```
/* Procedure which opens a new jwindow with a scrollpanel to chose an already created document
   It is called from the eventhandler by pressing the jbutton "oldb" */
setold:
        window~dispose()  -- Close the open jwindow
        CALL SysFileTree "C:\_old\*.*", "file", "O"     -- Creation of a
        -- stem(="array") of the system file tree of the specified folder

        Do i=1 To file.0
                PARSE VAR file.i prefix.i =9 filename.i       -- Extraction of the date
                -- information out of the full path of the file
        End

        -- Create java Swing objects
        label  = .JLabel~new("ÜBERSICHT HISTORISCHER DOKUMENTE");
        label2 = .JLabel~new("-----------------------------------------------");
        label3 = .JLabel~new("Bitte wählen Sie ein historisches Dokument aus!");
        window  = .JWindow~new();
        scrollp = .ScrollPane~new()
        jpanel  = .JPanel~new()
        jpanel2 = .JPanel~new()
        mainp   = .JPanel~new()
        jbutton1 = .JButton~new("OK");
        jbutton2 = .JButton~new("ABBRECHEN");
        buttonp = .JPanel~new();
        bevelborder = .BevelBorder~new(0)
        -- Set the Layout and add objects to the jpanels

        Do i=1 To file.0
                jpanel~setLayout(.GridLayout~new(i,2));
                jbutton.i = .JRadio~new(filename.i);
                jpanel~~add(jbutton.i)
                jbutton.i~bsf.addEventListener("action", "", t"=".true)
                jbutton.i~bsf.addEventListener("action", "", f"="i)
        End

        jpanel2~setLayout(.FlowLayout~new());
        jpanel2~~add(label)~~add(label2)~~add(label3)
        scrollp~~add(jpanel)  -- Add the jpanel to the scrollpanel
        buttonp~setLayout(.FlowLayout~new())
        buttonp~~add(jbutton1)~~add(jbutton2)
        mainp~setLayout(.GridLayout~new())
        mainp~add(jpanel2)~add(buttonp)


        -- Set the Gridlayout for the jwindow and add the jpanel and the scrollpanel
        window~getContentPane()~setLayout(.GridLayout~new(2,1));
        window~getContentPane()~setBorder(bevelborder);
        window~getContentPane()~~add(scrollp)~~add(mainp);
```

Figure 19: Code Snippet - Procedure setold

The next code snipped of figure 20 shows the calculation of the position of the jwindow like the first one before.

```
        -- Get the centered position of the jwindow depending on the screenresolution
        toolkit=.Toolkit~getDefaultToolkit()
        screenSize=Toolkit~getScreenSize()


        wx = screenSize~width()/3.4
        wy = screenSize~height()/3.4
        Parse var wx wx '.'
        Parse var wy wy '.'

        If wx<376 Then wx=376
        If wy<301 Then wy=301
        w_x = wx/2
        w_y = wy/2
        Parse var w_x w_x'.'
        Parse var w_y w_y'.'
        scx = screenSize~width()/2-w_x
        scy = screenSize~height()/2-w_y

        -- Set the location and size of the jwindow
        window~~setLocation(scx,scy)~~setSize(wx,wy)~~setVisible(.true)~~setAlwaysOnTop(.true)
        jbutton1~bsf.addEventListener("action", "", "Call open")
        jbutton2~bsf.addEventListener("action", "", "Call create_gui")
        jbutton2~bsf.addEventListener("action", "", "window~dispose()")

        /* A never ending loop which execute the messages from the eventhandler
           as a rexx program*/
        Do Forever
                event = bsf("pollEventText")
                interpret event
                Call set
        End

Exit
```

Figure 20: Code Snippet - Get position and set eventlistener

Procedure set is loaded by pressing a JRadioButton and is responsible for disabling all other JRadioButtons which may be selected so that only the current one is activated, shown in the code snippet of figure 21. This is achived using a “Do” loop and an “If” instruction. The procedure open is responsible for saving the filename of the historical document to a global variable and to execute the routine main_old which is the main routine for the loading of historical documents. At the end the shutdown procedure is used for closing the window by pressing the “ABBRECHEN” button within the JWindow.

```
set:

Do e=1 To file.0
          if jbutton.e <> jbutton.f Then jbutton.e~setSelected(.false);
End
return
Exit

open:
        window~dispose()                -- Close the open jwindow
        .local~filename =  filename.f
        if t=.true Then Call main_old
           Else Call setold             -- Start routine main_old
Exit


shutdown:
--window~dispose()                     -- Close the open jwindow
Exit
```

Figure 21: Code Snippet - Procedure set, open and shutdown

3.2.3 Routine main_new

The code snippet of figure 22 opens the template with the filename of the global variable which was retrieved before from the filesystem. First of all the Desktop service object of UNO is retrieved, with this object it is possible to get the XComponentLoader interface through the XDesktop interface. An URL is created with the makeUrl() method to set the complete path of the file. While automating OpenOffice.org arrays are used to set properties, therefore in this case an array is created and the new property value “Hidden” is inserted and set to true. The document is now opened in hidden mode with the use of the loadComponentFromURL() method of the XComponentLoader interface, so that the user gets no notice of any interaction.

After the creation of the document, the main interface is initialised, the access to the text is created by the getText() method. The model of the textdocument is retrieved by the initalisation of the XModel interface to be able to get a controller of the current model with the getCurrentController() method.

```
::Routine main_new
-- Main routine that opens a new template, save the content, export it to pdf and print it

        -- Open the newest template file
        oDesktop         = UNO.createDesktop()          -- Get the UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader -- Get the XcomponentLoader interface


        -- Open the document with the property value hidden
        url              = makeUrl("C:\_templates\".local~filename)  -- Get the template
        props            = bsf.createArray(.UNO~propertyValue,1)
        props[1]         = .UNO~PropertyValue~new
        props[1]~Name    = "Hidden"
        props[1]~Value   = box("boolean", .true)
        xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)
        -- Load the document with specified properties


        xWriterDocument  = xWriterComponent~XTextDocument -- Get the main interface
        xText            = xWriterDocument~getText()      -- Get the text of the document
        xModel           = xWriterDocument~XModel         -- Get the model from the textdocument
        xController      = xModel~getCurrentController()    -- Get the controller for the model
```

Figure 22: Code Snippet - Routine main_new

The controller is now needed to create a TextViewCursor which is used to move within the document, illustrated in figure 23. Additionally the Xsearchable interface is used with the included methodes createSearchDescriptor(),setSearchString(),findFirst() and findNext() among other ones for searching the document for special keywords. The search is supported through a “Do While” loop which continues the searching while keywords can be retrieved. The end of the loop is reached when a .nil object appears which is equivalent with the nonexistence of an object. The location of every found searchstring is saved to a stem variable.

```
-- With the help of the controller and XTextViewCursorSupplier we get the TextViewCursor
xViewCursorSupplier=xController~XTextViewCursorSupplier
xViewCursor = XViewCursorSupplier~getViewCursor

-- The XSearchable Interface gives the textdocument the possibility to search the content
xSearchabel = xWriterDocument~XSearchable
xSearchDescriptor = xSearchabel~createSearchDescriptor()
xSearchDescriptor~setSearchString("insert") -- Set the search string for the searchdescriptor
i=1
xFound = xSearchabel~findFirst(xSearchDescriptor) -- Find the first with the searchdescriptor
matching content
xFound.i = xFound

-- Loop which is searching for the searchstring until all matching strings are found
-- It saves the found strings to a stem (=array) variable
Do While xfound <> .nil
        .local~i=i
        i=i+1
        xFound = xSearchabel~findNext(xFound, xSearchDescriptor)
        xFound.i = xFound
End
i=i-1
```

Figure 23: Code Snippet - Interface initialisation

The next loop, displayed in figure 24, gets the information which is written after the searchstring, it is used to set the size, font and other aspects of the textfields which will be inserted. This gives the user of the template handling tool the possibility to change every aspect of the textfield without being forced to change the code of the macro script. For the retrieving of the information a TextCursor, a WordCursor, a SentenceCursor and a TextViewCursor is needed. First of all the found object of the Xsearchable interface initalises a XtextRange interface and gets the interface of the text in which the text position is stored with the getText() method. To get the position of the beginning of the retrieved TextRange the getStart() method is used. Now it is possible to create a TextCursor at the position of the found keyword with the help of the method createTextCursorByRange(). After that the TextViewCursor is moved to the position of the TextCursor and a WordCursor is created.

The Word Cursor is now used to jump from word to word by the method gotonextWord() and saves the information to variables of a stem using the two methods gotoendofWord(), used for marking the word, and getString() to get the string content. A SentenceCursor is needed because there is also information that is consisting of more than just one word, like the font. The procedure of retrieving the sentence is the same as getting a word out of the content, the only difference is that the method gotoendofSentence() is used to mark the sentence.

An “If” instruction saves the number of the textfield which contains the so called “Geschäftsnummer” to a global variable. The number is needed later on for the saving of the document because the name partial consists of that value. Finally the method getPosition() of the TextViewCursor is used for getting the exact position of the keyword within the template. This method returns a dimension object which includes the x and y coordinates of the position starting out from the upper left corner of the document. At the end of the searching process the document is closed.

```
-- Loop which gets the content that is positioned after the searchstring
        Do While i > 0
                -- Get the textrange of each searchstring
                xmodel      = xFound.i~XTextRange~getText()
                xposition   = xFound.i~XTextRange~getStart()
                xTextCursor = xmodel~createTextCursorByRange(xposition)
                xViewCursor~gotoRange(xTextCursor,.false)
                xWordCursor = xTextCursor~XWordCursor()
                xWordCursor~gotoendofWord(.true)
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                width.i=xWordCursor~getString()  -- Get the width of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                height.i=xWordCursor~getString() -- Get the height of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                text.i=xWordCursor~getString()     -- Get the text of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                f_height.i=xWordCursor~getString() -- Get the font height of the textbox
                xWordCursor~gotonextWord(.false)
                xSentenceCursor = xWordCursor~XSentenceCursor()
                xSentenceCursor~gotoendofSentence(.true)
                font.i=xSentenceCursor~getString() -- Get the font of the textbox
                if text.i = "Geschäftsnummer" then .local~number=i
                f.i=xViewCursor~getPosition()    -- Get the coordinates of the searchstring
                i=i-1
        End

    xWriterComponent~dispose()
```

Figure 24: Code Snippet – Loop for content extraction

After the previous steps the exact position of the document and the description of the textfields are retrieved. So the next big step is to create a new document which is based on the template and includes the textfields at the right position with the right formating as demonstrated in figure 25.

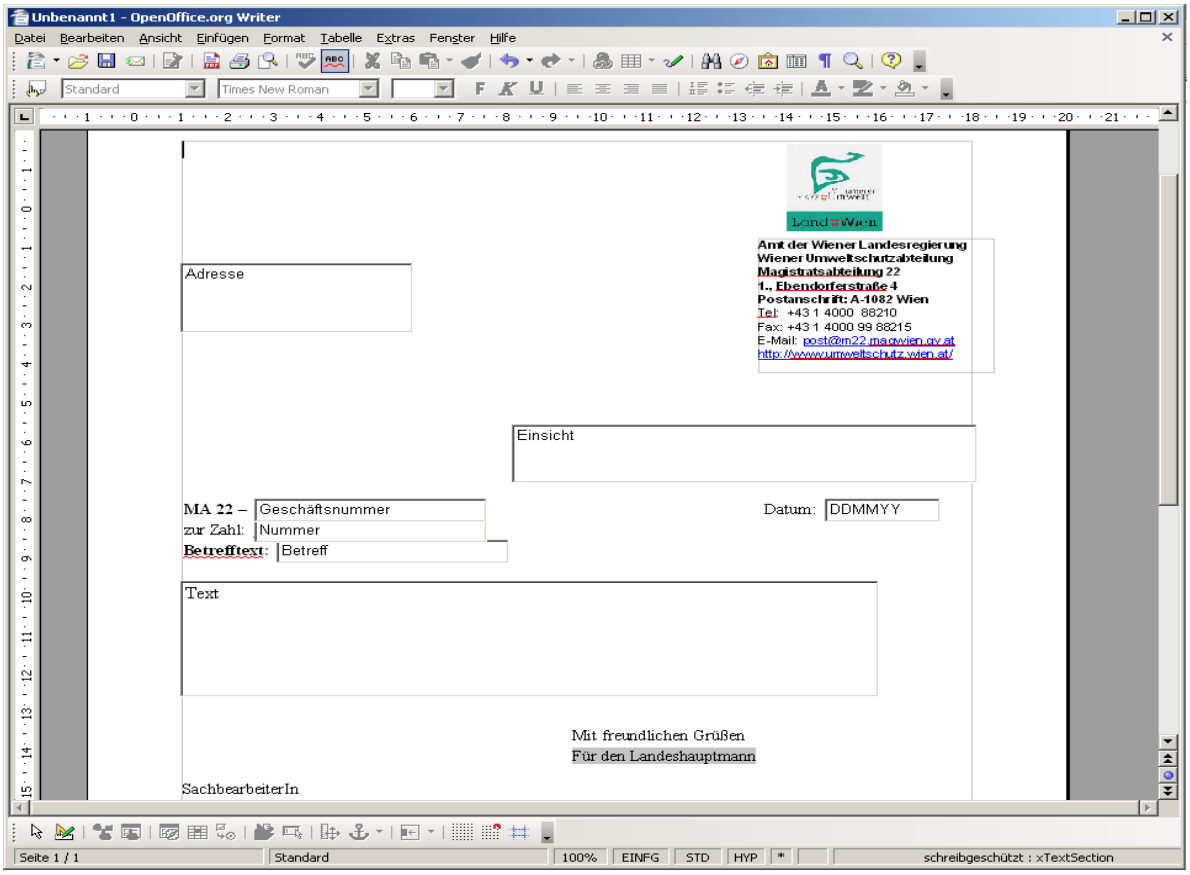

Figure 25: Sample document with inserted forms

First the XContext object is retrieved which is needed for getting the XMultiComponentFactory by the method getServiceManager(). The new document is created with the help of an URL, which describes the resource that should be loaded, which can be seen in figure 26. In our case we need an empty Writer component, the specified URL for it looks like this: "private:factory/swriter". In this case the method "loadComponentFromURL()" of the XcomponentLoader interface includes no properties because the new document need no specifications, which is realised with the setting ".UNO~noProps". The next step is to get the interface to the document as well as the service manager. All of the created objects are saved as ".local" variables because they will be needed also outside of the routine.

```
-- Open a new document to insert forms and content
xContext         = UNO.connect()  -- Connect to the server and retrieve the XContext object
xMCF             = xContext~getServiceManager       -- Get the XMultiComponentFactory
oDesktop         = UNO.createDesktop()              -- Create an UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- Get the componentLoader interface
from the desktop object

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

xWriterDocument  = xWriterComponent~XTextDocument
xText            = xWriterDocument~getText()
xServiceManager  = xWriterDocument~XMultiServiceFactory -- Get the service manager of the
textdocument

/* Save objects that are often used in the program as '.local' variables.
   With this they can be accessed like global variables in the program.  */
.local~Context         = xContext
.local~MCF             = xMCF
.local~ServiceManager  = xServiceManager
.local~WriterComponent = xWriterComponent
.local~Text            = xText
```

Figure 26: Code Snippet - New document creation

The next loop of figure 27 is used for the creation of the textfields, it runs while the variable i, which is the number of the found keywords, is bigger than zero. For the creation of the textfields the routine create_form is used which will be described in detail later on. The found parameters beside the keywords are assigned to each textfield with the help of this routine. For the description of the font a FontDescriptor has to be created using com.sun.star.awt.FontDescriptor. This descriptor includes structs that can be set with the message operator “~” of ooRexx like variables are initiated.

After the loop is finished the program creates a TextCursor which is saved as a “.local” variable. With the use of the XDocumentInsertable and the insertDocumentFromURL() method the content of the used template is added to the new document. The model and the controller of the document are retrieved to be able to create a TextViewCursor who as well is assigned to a “.local” variable.

```
i = .i
/* Loop that creates textboxes at the position of the searchstrings
   with the specified properties out of the content of the template.*/
Do While i>0
        textbox.i = create_form("TextField", width.i, height.i, f.i) -- Create a textbox
        textbox.i~setPropertyValue("Text", text.i)          -- Assign text to the textbox
        textbox.i~setPropertyValue("MultiLine", box("boolean", .true)) -- Set multiline
        -- Create a fontdescriptor
        FontDescriptor        = .bsf~new("com.sun.star.awt.FontDescriptor")
        FontDescriptor~Height = f_height.i
        FontDescriptor~Name   = font.i
        textbox.i~setPropertyValue("FontDescriptor", FontDescriptor) -- Assign the
        fontdescriptor to control the font
        i=i-1
End


xTextCursor       = xText~createTextCursor()     -- Create a TextCursor within the document
.local~TextCursor = xTextCursor
xTextCursor~XDocumentInsertable~insertDocumentFromURL(-
"file:///C:/_templates/".local~filename,.UNO~noProps)
xModel       = xWriterDocument~XModel         -- Get model from the textcomponent
xController  = xModel~getCurrentController()  -- Get the controller of the textdocument

-- Get the XViewCursor with the help of the controller and the XTextViewCursorSupplier
xViewCursorSupplier = xController~XTextViewCursorSupplier
xViewCursor         = XViewCursorSupplier~getViewCursor()
.local~ViewCursor   = xViewCursor
```

Figure 27: Code Snippet - Textfield insertion

After the loop is finished the program creates a TextCursor which is saved as a ".local" variable, shown in figure 28. With the use of the XDocumentInsertable and the insertDocumentFromURL() method the content of the used template is added to the new document. The model and the controller of the document are retrieved to be able to create a TextViewCursor who as well is assigned to a ".local" variable. Due to the reason that the keywords are still inside the textdocument the next loop is responsible for searching and deleting them with the use of the XReplaceable interface. The interface makes use of the createReplaceDescriptor() method to create the replace descriptor, and the setSearchString(), setReplaceString() and replaceAll() method to replace all matchings.

With the help of the CALL function the routines switch_design_mode, textsection and state_window are called. The switch_design_mode routine switches the shape control from design to live mode, the routine textsection creates a textsection and the routine state_window initialises a jwindow with information for the user. They will be discussed in detail later. At the end of this code part an "If" instruction checks if the "ABBRECHEN" button of the state_window JWindow was pressed and aborts if thats the case.

```
        xTextCursor       = xText~createTextCursor()     -- Create a TextCursor within the document
        .local~TextCursor = xTextCursor
        xTextCursor~XDocumentInsertable~insertDocumentFromURL(-
        "file:///C:/_templates/".local~filename,.UNO~noProps)
        xModel       = xWriterDocument~XModel         -- Get model from the textcomponent
        xController  = xModel~getCurrentController()  -- Get the controller of the textdocument

        -- Get the XViewCursor with the help of the controller and the XTextViewCursorSupplier
        xViewCursorSupplier = xController~XTextViewCursorSupplier
        xViewCursor         = XViewCursorSupplier~getViewCursor()
        .local~ViewCursor   = xViewCursor

        i=.i
        -- Loop which replaces all keywords of the template used for inserting textboxes
        Do While i > 0
                xReplaceable       = xWriterDocument~XReplaceable
                -- Create the replacedescriptor with the XReplaceable interface
                xReplaceDescriptor = xReplaceable~createReplaceDescriptor()
                xReplaceDescriptor~~setSearchString("insert "width.i" "height.i" "text.i"
                "f_height.i" "font.i)~~setReplaceString("") -- Set search and replace string
                xReplaceable~replaceAll(xReplaceDescriptor) -- Replace all strings matching the
                searchstring
                i=i-1
        End

Call switch_design_mode  -- Switch the design mode from design to live mode
Call textsection         -- Insert a protected textsection
Call state_window        -- Open the state info window of the macro

If .c<>true Then Call main_finish

Exit
```

Figure 28: Code Snippet - Content replace

After the initialising of the state_window routine the macro is in holding position until the user presses the “next” JButton inside the JWindow, illustrated in figure 29. When the button is activated the macro can continue and loads the main_finish procedure. This procedure is used to finish the new created document, it starts out with the saving of all content of the textfields to variables of a stem.

The macro finishes the document starting with the switching of the design mode of the shape to design and another loop is changing the border property value of all textfields to invisible and the content to readonly. This is needed for the export in pdf because the content would be able to be changed afterwards inside the PDF file and the borders of the textfields would also be visible.

```
/* Procedure that finalise the macro after the content is inserted
   and the user has pushed the "FORTSETZEN" button.*/
main_finish:
i=.i
-- Loop that gets all textcontent out of the textboxes
Do While i > 0
        tb.i = textbox.i~getPropertyValue("Text")
        i=i-1
End

Call switch_design_mode

i=.i
-- Loop which changes properties of the textboxes
Do While i > 0
        textbox.i~setPropertyValue("Border", box("short",0))  -- Set the border invisible
        textbox.i~setPropertyValue("ReadOnly", box("boolean",.true))  -- Set the content
        ReadOnly
        i=i-1
end
```

Figure 29: Code Snippet - Procedure main_finish

Now the document is exported into pdf using the XStorable interface, shown in figure 30. There is not much difference between saving a file into pdf or another format. The property value of the interface has to be set correctly using an array. The “Filtername” has to be set to “writer_pdf_Export” and the “CompressMode” must be set. The filename of the document consists of the current date and the “Geschäftsnummer”. The date can be get by using the built-in date() function of ooRexx and the number is retrieved with by getting access to the right textfield. The file is saved using the storeToUrl() method with the preset storing properties.

The next point is to print the document using the XPrintable interface. The method print() of the interface is using the property value .UNO~noProps so that the standard printer of the workstatioon is used. After that the RexxUtil function SysSleep is called to give OOo time for printing because the document is closed right afterwards.

```
        -- Set the storing properties
        xStorable = .WriterComponent~XStorable
        storeprops = bsf.createArray(.UNO~propertyValue, 3)
        storeprops[1] = .UNO~PropertyValue~new
        storeprops[1]~Name  = "FilterName"
        storeprops[1]~Value = "writer_pdf_Export"
        storeprops[2] = .UNO~PropertyValue~new
        storeprops[2]~Name  = "CompressMode"
        storeprops[2]~Value = 2

        xWriterDocument = .WriterComponent~XTextDocument  -- Loading of textdocument
        xText = xWriterDocument~getText()                  -- Get the content of the document

        date=date("S")
        e=.local~number
        file=date"_"tb.e

        .local~file=file

        xStorable~storeToUrl("file:///C:/"file".pdf", storeprops)  -- Store the document to the
        specified url

        xPrintable = xWriterComponent~XPrintable
        xPrintable~print(.UNO~noProps)

        CALL SysSleep 1


        xWriterComponent~dispose()       -- Closing of the Document
```

Figure 30: Code Snippet – Exporting and printing a document

The last important task of the routine is to save the content into a new textdocument without any properties, illustrated in figure 31.

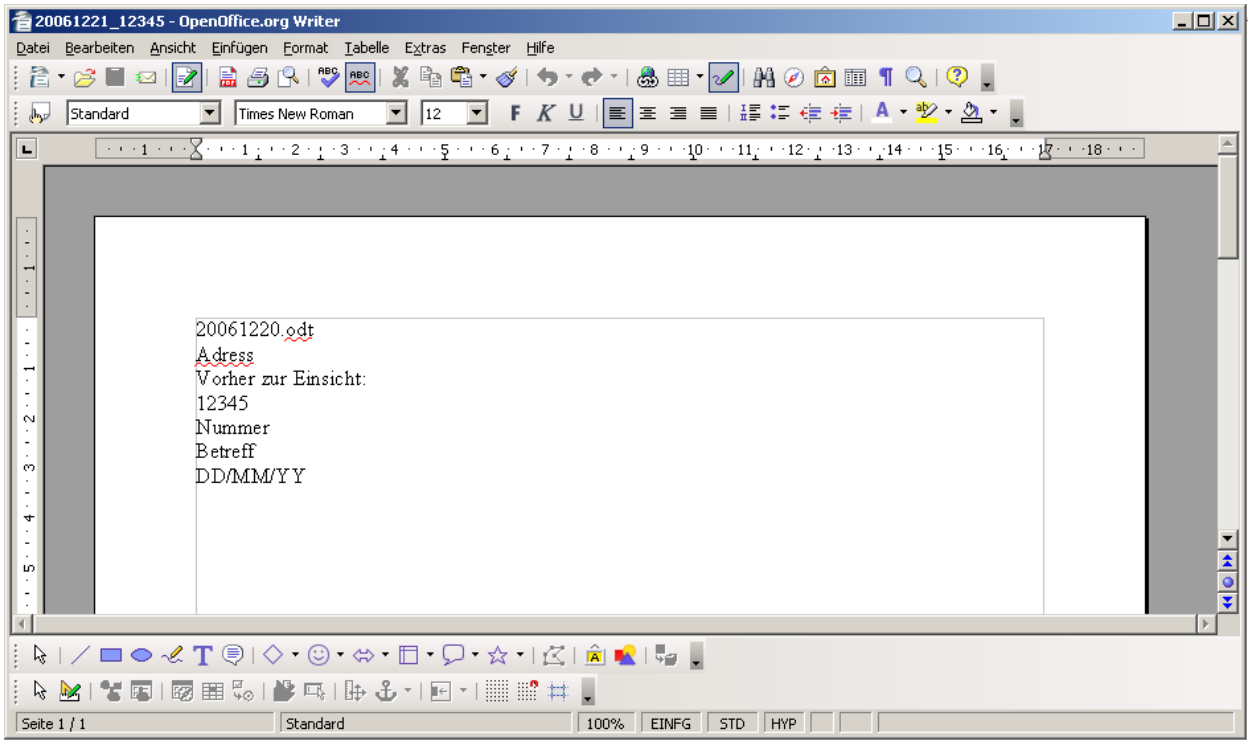

Figure 31: Sample content document

The first thing which is done is the creation of a new document illustrated in figure 32, the retrieving of the interface, the text and the creation of a TextCursor. Within the first line the name of the template which was used to create the document is inserted from a ".local" variable. Then a loop is creating a paragraphbreak for each textfield and inserts the content of them. When the loop is finished the document is stored to OpenOffice.org "odt" file without any properties, using the XStorable interface and the storeToUrl() method like before and closed after that.

```
oDesktop         = UNO.createDesktop()              -- Create an UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- Get the componentloader interface

 -- Open a new Writer document in hidden mode
 url              = "private:factory/swriter"
 props            = bsf.createArray(.UNO~propertyValue,1)
 props[1]         = .UNO~PropertyValue~new
 props[1]~Name    = "Hidden"
 props[1]~Value   = box("boolean", .true)
 xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)
 -- Load the document with specified properties

 xWriterDocument  = xWriterComponent~XTextDocument -- Get the main interface
 xText            = xWriterDocument~getText()      -- Get the text of the document
 xTextCursor = xText~createTextCursor()            -- Create a TextCursor


 xText~getEnd()~setstring(.filename)

 i=.i
 e=1
 -- Loop that insert the text of the textboxes and a paragraph breaks per textbox into the
    new document
 Do While e <= i
        xText~insertControlCharacter(xTextCursor,bsf.getConstant(-
        "com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)
        xText~getEnd()~setstring(tb.e)
        e=e+1
 End


 -- Store the created document to a specified location
 xStorable = xWriterComponent~XStorable
 xStorable~storeToUrl("file:///C:/_old/".file".odt", .UNO~noProps)
 xWriterDocument~dispose()
```

Figure 32: Code Snippet – Creation of a content file

At the end of the routine the user gets an information notification that the macro has finished and saved the file. This is done like the JWindows which were created before, the output is shown in figure 33. The program holds and is finished when the user presses the "Ok" JButton of the window.

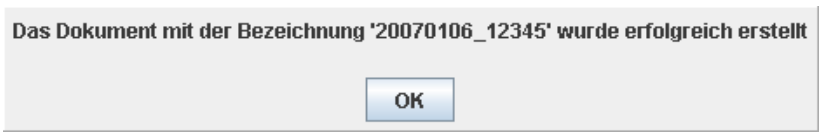

Figure 33: JWindow message box

The creation of the information JWindow is shown in figure 34 and is similary to the ones before.

```
-- MessageDialog who informs the user that the macro was completed successfully
window = .JWindow~new()
jpanel1 = .JPanel~new()
jpanel2 = .JPanel~new()
jlabel = .Jlabel~new(-
"Das Dokument mit der Bezeichnung '".file"' wurde erfolgreich erstellt")
jbutton     = .JButton~new("OK");
bevelborder = .BevelBorder~new(0)

jpanel1~~add(jlabel)
jpanel2~~add(jbutton)

window~getContentPane()~setLayout(.GridLayout~new(2,1));
window~getContentPane()~setBorder(bevelborder);
window~getContentPane()~~add(jpanel1)~~add(jpanel2)
window~~setLocationRelativeTo(window)~~setVisible(.true)~~setAlwaysOnTop(.true)

jbutton~bsf.addEventListener('action', '', 'call next')
Do Forever
        event = bsf("pollEventText")
        interpret event
End



next:

Exit
```

Figure 34: Code Snippet - Message box

3.2.4 Routine main_old

The second main routine is called main_old and is used for the merging of the content file and the template file which is illustrated in figure 35. Most of the script code of this routine equals the code of the main_new routine. Therefore in this chapter only the code parts which are different to the main_new routine are described.

The first part of the routine is different, it is used to load the document in hidden mode that the user has chosen within the graphical interface at the beginning. The name of the file was assigned to a ".local" variable which is used for the `makeUrl()` method to open the right document. Next a TextCursor, a WordCursor and a ParagraphCursor are initialised which are used to retrieve the content out of the document. As described before the content is structured into paragraphs, because of this matter the ParagraphCursor can be utilised to get each content of a paragraph. This is realised by a loop which jumps to the next paragraph with the method gotonextParagraph() of the XparagraphCursor interface

and marks the paragraph with the gotoendofParagraph() method. The WordCursor within the loop is needed to verify with the help of the isEndofWord() method if the loop has reached the end of the document. At the end all contents are saved to a stem and the document is closed. After this the retrieved template file is opened, which is done equally to routine main_new.

```
::Routine main_old  -- Routine responsible for loading old documents

        oDesktop         = UNO.createDesktop()           -- Create an UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader  -- Get the componentloader interface

        -- Open the content file of an old document in hidden mode
        url              = makeUrl("C:\_old\".filename)  -- Get the document
        props            = bsf.createArray(.UNO~propertyValue,1)
        props[1]         = .UNO~PropertyValue~new
        props[1]~Name    = "Hidden"
        props[1]~Value   = box("boolean", .true)
        xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

        xWriterDocument  = xWriterComponent~XTextDocument -- Get the main interface
        xText            = xWriterDocument~getText()      -- Get the text of the document


        xParagraphCursor~gotoendofParagraph(.true)
        .local~file = xParagraphCursor~getString()


        -- Loop that save each paragraph of the document to a variable
        a = 0
        i = 1
        Do While a = 0
                xParagraphCursor~gotonextParagraph(.false)
                a = xWordCursor~isEndofWord()
                xParagraphCursor~gotoendofParagraph(.true)
                text.i = xTextCursor~getString()
                i=i+1
        End

        xWriterComponent~dispose()  -- Close the document
```

Figure 35: Code Snippet - Routine main_old

The loop in figure 36 for getting the information used for the textfields is principally equally except of the text.i variable which is named text2.i and is not used for inserting text into the textfields, but only for the deleting of the keyword data. The reason for that is that the content is inserted from the data which was retrieved before out of the content document.

```
i=i-1
-- Loop which gets the content that is positioned after the searchstring
Do While i > 0
        xmodel      = xFound.i~XTextRange~getText() -- Get the textrange
        xposition   = xFound.i~XTextRange~getStart()
        xTextCursor = xmodel~createTextCursorByRange(xposition)
        xViewCursor~gotoRange(xTextCursor,.false)
        xWordCursor = xTextCursor~XWordCursor()

        xWordCursor~gotoendofWord(.true)
        xWordCursor~gotonextWord(.false)
        xWordCursor~gotoendofWord(.true)
        width.i=xWordCursor~getString()     -- Get the width of the textbox
        xWordCursor~gotonextWord(.false)
        xWordCursor~gotoendofWord(.true)
        height.i=xWordCursor~getString()    -- Get the height of the textbox
        xWordCursor~gotonextWord(.false)
        xWordCursor~gotoendofWord(.true)
        text2.i=xWordCursor~getString()     -- Get the text of the textbox
        xWordCursor~gotonextWord(.false)
        xWordCursor~gotoendofWord(.true)
        f_height.i=xWordCursor~getString()  -- Get the font height of the textbox
        xWordCursor~gotonextWord(.false)
        xSentenceCursor = xWordCursor~XSentenceCursor()
        xSentenceCursor~gotoendofSentence(.true)
        font.i=xSentenceCursor~getString()   -- Get the font of the textbox
        if text.i = "Geschäftsnummer" then .local~number=i

        f.i=xViewCursor~getPosition()        -- Get the coordinates of the searchstring
        i=i-1
End


xWriterComponent~dispose()  -- Close the document
```

Figure 36: Code Snippet - Loop for content extraction

The only difference between the loop in figure 37 and the one of the routine before is that the searchstring of the replacedescriptor includes text2.i in contrast to text.i.

```
i=.i
-- Loop which replaces all keywords of the template used for inserting textboxes
Do While i > 0
        xReplaceable       = xWriterDocument~XReplaceable
        xReplaceDescriptor = xReplaceable~createReplaceDescriptor() -- Create the
        replacedescriptor with the XReplaceable interface
        xReplaceDescriptor~~setSearchString("insert "width.i" "height.i" "text2.i"-
        "f_height.i" "font.i)~~setReplaceString("") -- Set search and replace string
        xReplaceable~replaceAll(xReplaceDescriptor) -- Replace all strings matching the
        searchstring
        i=i-1
End
```

Figure 37: Code Snippet - Loop for content replace

In this part, displayed in figure 38, the difference is that the state_window is not initialised. Furthermore the borders of the textfields are set to invisible and the text content is set to readonly before the user can make changes to the textfields. The reason for that is the user should not be able to change historical documents. Also there is no saving, exporting

and printing of the document.

```
        Call textsection         -- Insert a protected textsection

        i=.i
        -- Loop which changes properties of the textboxes
        Do While i > 0
                textbox.i~setPropertyValue("Border", box("short",0)) -- Set the border invisible
                textbox.i~setPropertyValue("ReadOnly", box("boolean",.true))  -- Set ReadOnly
                i=i-1
        End
        Call switch_design_mode  -- Switch the design mode from design to live mode
```

Figure 38: Code Snippet - Finalizing the historical document

3.2.5 Routine state_window

This routine is used to open a JWindow that is pausing the macro, the output can be seen in figure 39.

Figure 39: Info JWindow with control buttons

First of all like the other windows all of the objects are initialised and the layout of the JPanels and the window are set, the sourcecode is shown in figure 40. The button and the labels are added to the JPanels and they are added to the window .

```
::Routine state_window  -- Routine that opens a jwindow which is pausing the macro


        -- create java swing objects
        label    = .JLabel~new("")
        label2   = .JLabel~new("MAKRO ANGEHALTEN",0)
        label3   = .JLabel~new("-------------------------------",0)
        label4   = .JLabel~new("Bitte Inhalt eingeben!",0)
        label5   = .JLabel~new("Um Makro fortzusetzen,",0)
        label6   = .JLabel~new("bitte auf FORTSETZEN klicken!",0)
        window   = .JWindow~new();
        next     = .JButton~new("FORTSETZEN");
        cancel = .JButton~new("ABBRECHEN");
        icon     = .Icon~new("ma.png");
        jbuttonp = .JPanel~new();
        panel1   = .JPanel~new();
        panel2   = .JPanel~new();
        etchedborder = .EtchedBorder~new(0)


         -- set the layout for the jpanels and add objects
        jbuttonp~setLayout(.GridLayout~new(2,1))
        jbuttonp~~add(next)~~add(cancel);
        panel1~setLayout(.GridLayout~new(5,1))
        panel1~~add(label2)~~add(label3)~~add(label4)~~add(label5)~~add(label6)
        panel2~~add(label);


        -- Set the jwindow to use the GridBagLayout and add objects
        window~getContentPane()~setLayout(.GridBagLayout~new());
        window~getContentPane()~setBorder(etchedborder);
        window~getContentPane()~~add(panel1)~~add(jbuttonp)~~add(panel2);
```

Figure 40: Code Snippet - Routine state_window

The next step sets the size of the box like the JWindows before and calculates the exact position on the screen in the right lower corner above the task bar, shown in figure 41. This is accomplished by using the GraphicsEnvironment class which includes a method called getMaximumWindowBounds() that is able to calculate the screen resolution without the taskbar, so it is possible to set the window to the right position no matter what screen resolution or height of the taskbar is set. The width and the height can be retrieved by using width() and height() out of the GraphicsEnvironment object. The positon is set by substraction of the screen resolution minus the window size. Furthermore the location, size and visibility of the window is set, also an icon is added to a label of the window, like in the other examples described before.

```
-- Set the size of the jwindow
toolkit=.Toolkit~getDefaultToolkit()
screenSize = Toolkit~getScreenSize()

boxw = screenSize~width()/3
boxh = screenSize~height()/7
Parse var boxw boxw '.'
Parse var boxh boxh '.'


If boxw<426 Then boxw=426
If boxh<146 Then boxh=146
-- Get the Size of the screen without the task bar
ge = .GraphicsEnvironment~getLocalGraphicsEnvironment()
maximumWindowBounds = ge~getMaximumWindowBounds();
resx= maximumWindowBounds~width()
resy= maximumWindowBounds~height()

-- Get the position of the jwindow in the right corner of  the screen
boxpx = resx - boxw
boxp = resy - boxh


-- Set the location, size and visibility of the jwindow
window~~pack()~~setLocation(boxpx,boxp)~~setSize(boxw,boxh)-
~~setVisible(.true)~~setAlwaysOnTop(.true)

label~setIcon(icon);
```

Figure 41: Code Snippet - Setting the jwindow

Finally an eventlister is added to the JButton which closes the window by pressing the button and let the macro continue, illustrated in figure 42. The procedure next closes the JWindow and allows the script to continue and the cancel procedure let the user return to the main menu by closing the window and the document.

```
- Add eventhandling to the jbutton
next~bsf.addEventListener('action', '', 'Call next')
cancel~bsf.addEventListener('action', '', ".local~c= true")
cancel~bsf.addEventListener('action', '', 'Call cancel')


/* A never ending loop which execute the messages from the
   eventhandler as a rexx program */
Do Forever
        event = bsf("pollEventText")
        interpret event
End

Exit

next:
window~dispose()
Exit

cancel:
window~dispose()
.WriterComponent~dispose()
Call create_gui
Exit
```

Figure 42: Code Snippet - Eventlistener and procedures

3.2.6 Routine switch_design_mode

This routine is used to switch the design mode of the document shape from design to live mode, which can be seen in figure 43. Unfortunately OpenOffice.org provides no parameters for switching or triggering buttons, that is why in this case a dispatcher is needed to solve the task. First of all the com.sun.star.util.URL class is loaded with the BSF loadClass() method. An array is created which should contain one URL.class object, in this array a new object of the mentioned class is instanciated and to the element Complete of the struct URL the value .uno:SwitchControlDesignMode is assigned. According to the API of OpenOffice.org “Complete” which is a field value of the complete URL representation of a string.

The next task is to get the model of the document to be able to retrieve the current controller of the document. With the controller it is possible to get the XDispatchProvider interface. The method getFrame() from the XController interface is needed to get the current frame of the controller and the initialisation of the XDispatchProvider provides the dispatch provider of the frame.

```
::Routine switch_design_mode  -- Routine to switch the design mode from design to live mode
        url                    = bsf.loadClass("com.sun.star.util.URL") -- Load the class URL
        aToggleURL             = bsf.createArray(url, 1) -- Create an array to store the URL object
        aToggleURL[1]          = .bsf~new("com.sun.star.util.URL")
        aToggleURL[1]~Complete = ".uno:SwitchControlDesignMode"

        xmodel = .WriterComponent~XModel                        -- Get the model of  the document
        xController = xmodel~getCurrentController               -- Get the current controller
        xDispatchProvider = xController~getFrame~XDispatchProvider   -- Get the dispatch provider
```

Figure 43: Code Snippet - Routine switch_design_mode

Now with the use of the XmultiComponentFactory a new instance of an URLTransformer is initialised by the method createInstanceWithContext() which is illustrated in figure 44. This component supports the services of the current factory. Because the URL which was created before has to be parsed the XURLTransformer interface of the URLTransformer service is created. With the support of this interface it is possible to parse the string of the url to a syntactically correct URL with the method parseStrict().

After the url is complete the XDispatchProvider uses the queryDispatch() method to get the dispatch object. The XDispatch object supports the interface XDispatch which provides

among others the method dispatch(). This method is used to execute the parsed URL asynchronously.

```
        -- Use an URLTransformer to parse the url
        frameDesktop = .MCF~createInstanceWithContext("com.sun.star.util.URLTransformer",.Context)
        xURLTransformer = frameDesktop~XURLTransformer  -- Initialisation of the interface
        xURLTransformer~parseStrict(aToggleURL)         -- Parse the url


        -- Get all dispatcher from the specified url
        xDispatcher = xDispatchProvider~queryDispatch( aToggleURL[1],"", 0 )
        xDispatcher~dispatch( aToggleURL[1], .UNO~noProps ) -- Execute a dispatcher from the url
```

Figure 44: Code Snippet - URL dispatching

3.2.7 Routine textsection

The routine textsection is used to protect the document against changes, shown in figure 45. The TextViewCursor, which was saved before to a ".local" variable, marks the whole document as a single TextRange. The global available Servicemanager creates a new instance of the com.sun.star.text.TextSection to which a name has to be assigned through the service com.sun.star.container.XNamed and the method setName(). Furthermore the property IsProtected of the textsection is set to the value "true" using the already known XPropertySet() interface. After the object is prepared for the insertion the XTextContent() interface is called to enable the textsection object so that it can be inserted into the text. Finally the section is added to the TextCursor with the method insertTextContent().

```
:Routine textsection  -- Routine to insert a protected textsection accross the whole document
        .ViewCursor~gotoStart(.false)
        .TextCursor~gotoEnd(.true)      -- Mark the whole document as a TextRange
        -- Create an instance of the textsection
        xTextSection = .ServiceManager~createInstance("com.sun.star.text.TextSection")
        xTextSection~XNamed~setName("xTextSection")    -- Set a name for the textsection
        xTextSectionprops = xTextSection~XPropertySet()
        xTextSectionprops~setPropertyValue("IsProtected",box("boolean",.true)) -- Set the property
        TextSection = xTextSection~XTextContent()
        .Text~insertTextContent(.TextCursor,TextSection,.true) -- Insert the textsection
```

Figure 45: Code Snippet - Routine textsection

3.2.8 Routine create_form

The last routine of the macro script is the create_form shown in figure 46 which is used by the two main routines for the creation of the shape and the insertion of the textfields. First the global XWriterComponent is used to get a XMultiServiceFactory which is able to create an instance of the com.sun.star.drawing.ControlShape. This service makes usage of the XControlShape interface to get access to the control model.

In our case values must be assigned from outside of this routine which are needed to create form components. These values include the name of the component, the size and the location within the shape. To make this possible the built-in argument function ARG() of ooRexx is used, a number for each value is inserted into this function to create an array. Each function is set at the exact place where the value is needed. Now the routine is able to be initialised with assigned values from outside. The code example in figure 46 of the main routine shows how it is realised by calling create_form().

```
::Routine create_form -- Routine to create and initialize a shape for the document
        xMSF = .WriterComponent~XMultiServiceFactory
        xControlShape = xMSF~createInstance("com.sun.star.drawing.ControlShape")~XControlShape
        -- Create a control shape

        sQualifiedComponentName = "com.sun.star.form.component."ARG(1)  -- Create a form component
        xControlModel =.MCF~createInstanceWithContext(-
        sQualifiedComponentName,.Context)~XControlModel -- Create a control model


        textbox.i = create_form("TextField", width.i, height.i, f.i)
```

Figure 46: Code Snippet - Routine create_form

The width and height of the form are set by the values which are retrieved out of the content of the template and then assigned to the routine, illustrated in figure 47. Therefore for setting the size a new com.sun.star.awt.Size object has to be created using the BSF function new(). The positon is retrieved from the location of the TextViewCursor which returns a dimension object, therefore only one argument function is needed. Furthermore the anchortype is set to AT_PARAGRAPH and with the method setcontrol of the Xcontrolshape interface the created control model is set to the current shape.

```
        -- Set the position and size of the form
        xControlShape~setSize(.bsf~new("com.sun.star.awt.Size", ARG(2)*100, ARG(3)*100))
        -- The size is specified in 100th/mm

        xControlShape~setPosition(ARG(4))
        xPropertySet = xControlShape~XPropertySet       -- Get the propertyset of the shape
        xPropertySet~setPropertyValue("AnchorType", bsf.getConstant(-
        "com.sun.star.text.TextContentAnchorType", "AT_PARAGRAPH"))    -- Adjust the anchor to the
        paragraph

        xControlShape~setControl(xControlModel) -- Set the ControlModel of the shape
```

Figure 47: Code Snippet - Setting the xControlshape

The last step is to create a XDrawPageSupplier, to get the current page with the method getDrawPage(), to initialise the XShapes interface and then add the control shape to the shapes collection of the textdocument. At the end the command **return** is inserted to give the possibility of adjustments to the properties of the control model shown in figure 48.

```
        -- Add the shape to the shapes of the document
        xDrawPageSupplier = .WriterComponent~XDrawPageSupplier
        xDrawPage         = xDrawPageSupplier~getDrawPage()
        xShapes           = xDrawPage~XShapes
        xShapes~add(xControlShape)

        return xControlModel~XPropertySet
        -- Returns the XPropertySet interface to give the possibility of adjustments
```

Figure 48: Code Snippet - Adding the shape

4 Conclusion

The Resumee to the automation process of OpenOffice.org with the help of ooRexx is quite positive. Despite the fact that I had never worked with ooRexx BSF4Rexx and the OpenOffice.org API before, it was apart from a steep learning process at the beginning not very difficult to get used to the syntax of ooRexx which is quite easy to learn if you are used to any programming languages. Once the main concepts of this scripting language are understood the use of the language is quite intuitive which results mainly from the humanlike syntax and the few rules. Additionally it must be mentioned that the project could be finished positively within the given time frame. Nevertheless not the whole programming phase was positiv, there appeared also some problems. Most of them occured because the syntax had to be translated from Java to ooRexx. The reason for this is that most of the examples out of the developers guide of OpenOffice.org are written in ooBasic or Java. Also there exist not many code examples of ooRexx in combination with OpenOffice.org, so in our case Java was very comfortable to be translated into the ooRexx syntax because there exist many similarities.

The main part of the problems resulted out of the API of OpenOffice.org which provides aside from the well documented services, interfaces and properties not enough code snippet examples to give developers a detailed overview how to use the offered functionality. OpenOffice.org is offering indeed examples in different programming language including ooRexx on the so called codesnippetbase website and there exists also other sources like the OpenOffice.org forum where users of the API are helping each other to solve problems. However in my opinion the amount of provided code is not enough to support developers sufficiently. Due to these problems many problems had to be solved by trial and error which was at the beginning very time consuming.

From my point of view ooRexx is very proper for the automation of OpenOffice.org with the simplicity of an script language like ooBasic and the powerful possibilities of Java. It is quite possible that ooRexx will become more important in respect of OpenOffice.org dependent that more documentation sources for the automation with ooRexx will be provided to support developers.

5 List of References

[AHAM05] Andreas Ahammer: Bachelor Course Paper
OpenOffice.org Automation: Object Model, Scripting Languages, "Nutshell"-Examples
Department of Business Informatics (Prof. Dr. Rony G. Flatscher)
Vienna University of Economics and Business Administration
2006-11-06

[BURG06] Martin Burger: Bachelor Course Paper
OpenOffice.org Automatisation with Object Rexx
Department of Business Informatics (Prof. Dr. Rony G. Flatscher)
Vienna University of Economics and Business Administration
2006-05-19

[FLAT05] Flatscher, Rony G.: Automating OpenOffice.org With OOREXX Architecture, Gluing To Rexx Using BSF4Rexx
http://wi.wu-wien.ac.at/rgf/rexx/orx16/2005_orx16_Gluing2ooRexx_OOo.pdf
retrieved on 2006-11-15

[FLAT06-1] Flatscher, Rony G.: Resurrecting REXX, Introducing Object Rexx
http://prog.vub.ac.be/~wdmeuter/RDL06/Flatscher.pdf
retrieved on 2006-10-25

[FLAT06-2] Flatscher, Rony G.: The Vienna Version of BSF4Rexx
http://wi.wu-wien.ac.at/rgf/rexx/orx17/2006_orx17_BSF_ViennaEd.pdf
retrieved on 2006-11-14

[OOAP06] OpenOffice.org API: TextCursor
http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextCursor.html
retrieved on 2006-11-03

[OORE06] ooRexx: About Open Object Rexx
http://www.oorexx.org/index.html
retrieved on 2006-11-05

[OPEN05] OpenOffice.org: OpenOffice.org 2.0 Developer's Guide
http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf
retrieved on 2006-11-01

[OPEN06] OpenOffice.org: About Us: OpenOffice.org
http://about.openoffice.org/index.html
retrieved on 2006-11-20

[PITO06] Pitonyak, Andrew: Useful Macor Information For OpenOffice
http://www.pitonyak.org/AndrewMacro.pdf
retrieved on 2006-11-05

[PITO04] Pitonyak, Andrew: OpenOffice.org Macros Explained
Hentzenwerke Publishing, July 2004
retrieved on 2006-11-29

[PREM06] Matthias Prem: Bachelor Course Paper
ooRexx Snippets for OpenOffice.org Writer
Department of Business Informatics (Prof. Dr. Rony G. Flatscher)
Vienna University of Economics and Business Administration
2006-07-24

6 Appendix

6.1 Sourcecode

```
/*****************************************************************************/
/************************TEMPLATE HANDLING TOOL*****************************/
/*****************************by Michael Kauril*******************************/



-- Import of all needed Java classes
.bsf~bsf.import("java.awt.GridLayout","GridLayout")
.bsf~bsf.import("java.awt.GridBagLayout","GridBagLayout")
.bsf~bsf.import("java.awt.Toolkit","Toolkit")
.bsf~bsf.import("java.awt.ScrollPane","ScrollPane")
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout")
.bsf~bsf.import("java.awt.GraphicsEnvironment", "GraphicsEnvironment")
.bsf~bsf.import("javax.swing.JFrame","JFrame")
.bsf~bsf.import("javax.swing.JLabel","JLabel")
.bsf~bsf.import("javax.swing.JRadioButton","JRadio")
.bsf~bsf.import("javax.swing.ImageIcon","Icon")
.bsf~bsf.import("javax.swing.JWindow","JWindow")
.bsf~bsf.import("javax.swing.JPanel","JPanel")
.bsf~bsf.import("javax.swing.JButton","JButton")
.bsf~bsf.import("javax.swing.JOptionPane", "JOptionPane")
.bsf~bsf.import("javax.swing.border.BevelBorder", "BevelBorder")
.bsf~bsf.import("javax.swing.border.EtchedBorder", "EtchedBorder")


Call create_gui



::Requires BSF.CLS        -- Make oo-like BSF4Rexx support available
::Requires UNO.CLS        -- Get UNO support
::Requires RXREGEXP.CLS   -- Get the support of regular expressions
::Routine makeUrl         -- operating system independent
Return  ConvertToURL(stream(arg(1), "c", "query exists"))
Exit



/*****************************************************************************/
-------------------------------ROUTINES--------------------------------------
/*****************************************************************************/


/**************************ROUTINE create_gui*********************************/

::Routine create_gui
-- Create a java swing menu with radiobuttons
-- Create java swing objects
label   = .JLabel~new("BITTE TREFFEN SIE EINE AUSWAHL:");
label2  = .JLabel~new("-----------------------------------------------------")

label3  = .JLabel~new("Klicken Sie 'AKTUELL' zum Erstellen eines neuen Dokumentes");
label4  = .JLabel~new("Klicken Sie 'HISTORISCH' zum Laden eines alten Dokumentes");
window  = .JWindow~new();
labelp  = .JPanel~new();
buttonp1  = .JPanel~new();
jbutton1 = .JButton~new("AKTUELL");
jbutton2 = .JButton~new("HISTORISCH");
jbutton3 = .JButton~new("ABBRECHEN");

buttonp2 = .JPanel~new();
bevelborder = .BevelBorder~new(0)

-- Set the Layout and add objects to jpanels
labelp~setLayout(.FlowLayout~new())
labelp~~add(label)~~add(label2)~~add(label3)~~add(label4)
buttonp1~setLayout(.FlowLayout~new())
buttonp1~~add(jbutton1)~~add(jbutton2)
buttonp2~setLayout(.FlowLayout~new())
buttonp2~~add(jbutton3)


-- Set the position of the jwindow to the center of the screen
toolkit=.Toolkit~getDefaultToolkit()
```

```
screenSize = Toolkit~getScreenSize()

wx = screenSize~width()/3.2
wy = screenSize~height()/4
Parse var wx wx '.'
Parse var wy wy '.'

If wx<400 then wx=400
If wy<256 then wy=256

w_x = wx/2
w_y = wy/2
Parse var w_x w_x'.'
Parse var w_y w_y'.'
scx = screenSize~width()/2-w_x
scy = screenSize~height()/2-w_y

-- Set the layout of the jwindow and add the jpanels
window~getContentPane()~setLayout(.GridLayout~new(3,1));
window~getContentPane()~setBorder(bevelborder);
window~getContentPane()~~add(labelp)~~add(buttonp1)~~add(buttonp2);

-- Set the location, size and visibility of the jwindow
window~~setLocation(scx,scy)~~setSize(wx,wy)~~setVisible(.true)-
~~setAlwaysOnTop(.true)

-- Add eventlisteners to the jbuttons
jbutton1~bsf.addEventListener("action", "", "Call setnew")
jbutton2~bsf.addEventListener("action", "", "Call setold")
jbutton3~bsf.addEventListener("action", "", "Exit")


/* A never ending loop which execute the messages from the
   eventhandler as a rexx program*/
Do forever
        event = bsf("pollEventText")
        interpret event
End

Exit


/* Procedure which gets the latest template and opens it
   It is called from the eventhandler by pressing the jbutton "newb" */

setnew:
        window~dispose()  -- Close the open jwindow
        CALL SysFileTree "C:\_templates\*.*", "file", "O" -- Creation of a
        -- stem(="array") of the system file tree of the specified folder

        Do i=1 TO file.0
                PARSE VAR file.i prefix.i =15 filename.i =23 subfix.i -- Extraction
                -- of the date information out of the full path of the file
        End

        i=2
        e=1
        f=0

        -- Comparison of the Dates to find out the latest one
        Do Until i > file.0
                If filename.e>filename.i Then e=e-1;
                If filename.e>filename.i Then f=e;
                Else f=i;
                e=e+1
                i=i+1
        End

        .local~filename =  filename.f""subfix.f
        Call main_new
Exit

/* Procedure which opens a new jwindow with a scrollpanel to chose an already created document
   It is called from the eventhandler by pressing the jbutton "oldb" */
setold:
        window~dispose()  -- Close the open jwindow
        CALL SysFileTree "C:\_old\*.*", "file", "O"     -- Creation of a
        -- stem(="array") of the system file tree of the specified folder

        Do i=1 To file.0
                PARSE VAR file.i prefix.i =9 filename.i       -- Extraction of the date
                -- information out of the full path of the file
        End
```

```
        -- Create java Swing objects
        label  = .JLabel~new("ÜBERSICHT HISTORISCHER DOKUMENTE");
        label2 = .JLabel~new("-----------------------------------------------");
        label3 = .JLabel~new("Bitte wählen Sie ein historisches Dokument aus!");
        window  = .JWindow~new();
        scrollp = .ScrollPane~new()
        jpanel  = .JPanel~new()
        jpanel2 = .JPanel~new()
        mainp   = .JPanel~new()
        jbutton1 = .JButton~new("OK");
        jbutton2 = .JButton~new("ABBRECHEN");
        buttonp = .JPanel~new();
        bevelborder = .BevelBorder~new(0)
        -- Set the Layout and add objects to the jpanels

        Do i=1 To file.0
                jpanel~setLayout(.GridLayout~new(i,2));
                jbutton.i = .JRadio~new(filename.i);
                jpanel~~add(jbutton.i)
                jbutton.i~bsf.addEventListener("action", "", t"=".true)
                jbutton.i~bsf.addEventListener("action", "", f"="i)
        End

        jpanel2~setLayout(.FlowLayout~new());
        jpanel2~~add(label)~~add(label2)~~add(label3)
        scrollp~~add(jpanel)  -- Add the jpanel to the scrollpanel
        buttonp~setLayout(.FlowLayout~new())
        buttonp~~add(jbutton1)~~add(jbutton2)
        mainp~setLayout(.GridLayout~new())
        mainp~add(jpanel2)~add(buttonp)


        -- Set the Gridlayout for the jwindow and add the jpanel and the scrollpanel
        window~getContentPane()~setLayout(.GridLayout~new(2,1));
        window~getContentPane()~setBorder(bevelborder);
        window~getContentPane()~~add(scrollp)~~add(mainp);

        -- Get the centered position of the jwindow depending on the screenresolution
        toolkit=.Toolkit~getDefaultToolkit()
        screenSize=Toolkit~getScreenSize()


        wx = screenSize~width()/3.4
        wy = screenSize~height()/3.4
        Parse var wx wx '.'
        Parse var wy wy '.'

        If wx<376 Then wx=376
        If wy<301 Then wy=301
        w_x = wx/2
        w_y = wy/2
        Parse var w_x w_x'.'
        Parse var w_y w_y'.'
        scx = screenSize~width()/2-w_x
        scy = screenSize~height()/2-w_y

        -- Set the location and size of the jwindow
        window~~setLocation(scx,scy)~~setSize(wx,wy)~~setVisible(.true)~~setAlwaysOnTop(.true)
        jbutton1~bsf.addEventListener("action", "", "Call open")
        jbutton2~bsf.addEventListener("action", "", "Call create_gui")
        jbutton2~bsf.addEventListener("action", "", "window~dispose()")

        /* A never ending loop which execute the messages from the eventhandler
           as a rexx program*/
        Do Forever
                event = bsf("pollEventText")
                interpret event
                Call set
        End

Exit

set:

Do e=1 To file.0
          if jbutton.e <> jbutton.f Then jbutton.e~setSelected(.false);
End
return
Exit


open:
```

```
        window~dispose()                -- Close the open jwindow
        .local~filename =  filename.f
        if t=.true Then Call main_old
           Else Call setold             -- Start routine main_old
Exit

shutdown:
--window~dispose()                     -- Close the open jwindow
Exit



/*****************************ROUTINE main_new*******************************/

::Routine main_new  -- Main routine that opens a new template, save the content,
                    -- export it to pdf and print it

        -- Open the newest template file
        oDesktop         = UNO.createDesktop() -- Get the UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader -- Get the XComponent-
                                                              -- Loader interface

        -- Open the document with the property value hidden
        url              = makeUrl("C:\_templates\".local~filename)
        -- Get the template from the specified folder
        props            = bsf.createArray(.UNO~propertyValue,1)
        props[1]         = .UNO~PropertyValue~new
        props[1]~Name    = "Hidden"
        props[1]~Value   = box("boolean", .true)
        xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)
        -- Load the document with specified properties

        xWriterDocument  = xWriterComponent~XTextDocument -- Get the main interface of the textdocument
        xText            = xWriterDocument~getText()      -- Get the text of the document
        xModel           = xWriterDocument~XModel        -- Get the model from the textdocument
        xController      = xModel~getCurrentController    -- Get the controller for the model

        -- With the help of the controller and XTextViewCursorSupplier we get the TextViewCursor
        xViewCursorSupplier=xController~XTextViewCursorSupplier
        xViewCursor = XViewCursorSupplier~getViewCursor

        -- The XSearchable Interface gives the textdocument the possibility to search the content
        xSearchabel = xWriterDocument~XSearchable
        xSearchDescriptor = xSearchabel~createSearchDescriptor()
        xSearchDescriptor~setSearchString("insert") -- Set the search string for the searchdescriptor
        i=1
        xFound = xSearchabel~findFirst(xSearchDescriptor) -- Find the first matching content
        xFound.i = xFound

        /* Loop which is searching for the searchstring until all matching strings
           are found. It saves the found strings to a stem (=array) variable */
        Do While xfound <> .nil
                .local~i=i
                i=i+1
                xFound = xSearchabel~findNext(xFound, xSearchDescriptor)
                xFound.i = xFound
        End

        i=i-1

        -- Loop which gets the content that is positioned after the searchstring
        Do While i > 0
                xmodel      = xFound.i~XTextRange~getText() /* Get the textrange
                                                               of each searchstring */
                xposition   = xFound.i~XTextRange~getStart()
                xTextCursor = xmodel~createTextCursorByRange(xposition)
                xViewCursor~gotoRange(xTextCursor,.false)
                xWordCursor = xTextCursor~XWordCursor()

                xWordCursor~gotoendofWord(.true)
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                width.i=xWordCursor~getString() -- Get the width of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                height.i=xWordCursor~getString() -- Get the height of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                text.i=xWordCursor~getString()   -- Get the text of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                f_height.i=xWordCursor~getString() -- Get the font height of the textbox
                xWordCursor~gotonextWord(.false)
```

```
        xSentenceCursor = xWordCursor~XSentenceCursor()
        xSentenceCursor~gotoendofSentence(.true)
        font.i=xSentenceCursor~getString() -- Get the font of the textbox
        if text.i = "Geschäftsnummer" then .local~number=i

        f.i=xViewCursor~getPosition()  -- Get the coordinates of the searchstring
        i=i-1
End

xWriterComponent~dispose()




-- Open a new document to insert forms and content

xContext         = UNO.connect()
-- Get a connection to the server and retrieve the XContext object
xMCF             = xContext~getServiceManager
-- Get the XMultiComponentFactory
oDesktop         = UNO.createDesktop()
-- Create an UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader
-- Get the componentLoader interface from the desktop object


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

xWriterDocument  = xWriterComponent~XTextDocument
xText            = xWriterDocument~getText()
xServiceManager = xWriterDocument~XMultiServiceFactory
-- Get the service manager of the textdocument

/* Save objects that are often used in the program as '.local' variables.
   With this they can be accessed like global variables in the program.  */
.local~Context         = xContext
.local~MCF             = xMCF
.local~ServiceManager  = xServiceManager
.local~WriterComponent = xWriterComponent
.local~Text            = xText


i = .i
/* Loop that creates textboxes at the position of the searchstrings
   with the specified properties out of the content of the template.*/
Do While i>0
        textbox.i = create_form("TextField", width.i, height.i, f.i)
        -- Create a textbox with the routine 'create_form'
        textbox.i~setPropertyValue("Text", text.i)
        -- Assign text to the textbox
        textbox.i~setPropertyValue("MultiLine", box("boolean", .true))
        -- Set the textbox multiline
        FontDescriptor        = .bsf~new("com.sun.star.awt.FontDescriptor")
        -- Create a fontdescriptor
        FontDescriptor~Height = f_height.i
        FontDescriptor~Name   = font.i
        textbox.i~setPropertyValue("FontDescriptor", FontDescriptor)
        -- Assign the fontdescriptor to control the font
        i=i-1
End


xTextCursor       = xText~createTextCursor() -- Create a TextCursor within the document
.local~TextCursor = xTextCursor
xTextCursor~XDocumentInsertable~insertDocumentFromURL(-
"file:///C:/_templates/".local~filename,.UNO~noProps)
xModel            = xWriterDocument~XModel  -- Get model from the textcomponent
xController       = xModel~getCurrentController()
-- Get the controller of the textdocument

-- Get the XViewCursor with the help of the controller and the XTextViewCursorSupplier
xViewCursorSupplier = xController~XTextViewCursorSupplier
xViewCursor         = XViewCursorSupplier~getViewCursor()
.local~ViewCursor   = xViewCursor

i=.i
-- Loop which replaces all keywords of the template used for inserting textboxes
Do While i > 0
        xReplaceable       = xWriterDocument~XReplaceable
```

```
        xReplaceDescriptor = xReplaceable~createReplaceDescriptor()
        -- Create the replacedescriptor with the XReplaceable interface
        xReplaceDescriptor~~setSearchString("insert "width.i" "height.i"-"-
        text.i" "f_height.i" "font.i)~~setReplaceString("")
        -- Set search and replace string
        xReplaceable~replaceAll(xReplaceDescriptor)
        -- Replace all strings matching the searchstring
        i=i-1
End

Call switch_design_mode  -- Switch the design mode from design to live mode
Call textsection         -- Insert a protected textsection
Call state_window        -- Open the state info window of the macro

If .c<>true Then Call main_finish

Exit




/* Procedure that finalise the macro after the content is inserted
   and the user has pushed the "FORTSETZEN" button.*/
main_finish:
i=.i
-- Loop that gets all textcontent out of the textboxes
Do While i > 0
        tb.i = textbox.i~getPropertyValue("Text")
        i=i-1
End

Call switch_design_mode

i=.i
-- Loop which changes properties of the textboxes
Do While i > 0
        textbox.i~setPropertyValue("Border", box("short",0))
        -- Set the border invisible
        textbox.i~setPropertyValue("ReadOnly", box("boolean",.true))
        -- Set the content ReadOnly
        i=i-1
end

/*******************Saving the document as PDF************************/

-- Set the storing properties
xStorable = .WriterComponent~XStorable
storeprops = bsf.createArray(.UNO~propertyValue, 3)
storeprops[1] = .UNO~PropertyValue~new
storeprops[1]~Name  = "FilterName"
storeprops[1]~Value = "writer_pdf_Export"
storeprops[2] = .UNO~PropertyValue~new
storeprops[2]~Name  = "CompressMode"
storeprops[2]~Value = 2

xWriterDocument = .WriterComponent~XTextDocument  -- Loading of textdocument
xText = xWriterDocument~getText()     -- Get the content of the document

date=date("S")
e=.number
file=date"_"tb.e

.local~file=file

xStorable~storeToUrl("file:///C:/"file".pdf", storeprops)
-- Store the document to the specified url


/*******************Printing of the document**************************/

xPrintable = xWriterComponent~XPrintable
xPrintable~print(.UNO~noProps)

CALL SysSleep 1


xWriterComponent~dispose()   -- Closing of the Document
```

```
        /***************Saving the content of the document******************/

        oDesktop         = UNO.createDesktop() -- Create an UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader
        -- Get the componentloader interface from the desktop object

        -- Open a new Writer document in hidden mode
        url              = "private:factory/swriter"
        props            = bsf.createArray(.UNO~propertyValue,1)
        props[1]         = .UNO~PropertyValue~new
        props[1]~Name    = "Hidden"
        props[1]~Value   = box("boolean", .true)
        xWriterComponent = xComponentLoader~loadComponentFromURL(-
        url, "_blank", 0, props) -- Load the document with specified properties

        xWriterDocument  = xWriterComponent~XTextDocument /* Get the main interface
                                                             of the textdocument */
        xText            = xWriterDocument~getText()      /* Get the text of the
                                                             document */
        xTextCursor = xText~createTextCursor()            /* Create a TextCursor
                                                             within the document */

        xText~getEnd()~setstring(.filename)

        i=.i
        e=1
        /* Loop that insert the text of the textboxes and a paragraph
           breaks per textbox into the new document */
        Do While e <= i
                xText~insertControlCharacter(xTextCursor,bsf.getConstant(-
                "com.sun.star.text.ControlCharacter","PARAGRAPH_BREAK"),.false)
                xText~getEnd()~setstring(tb.e)
                e=e+1
        End

        -- Store the created document to a specified location
        xStorable = xWriterComponent~XStorable
        xStorable~storeToUrl("file:///C:/_old/".file".odt", .UNO~noProps)
        xWriterDocument~dispose()

        -- MessageDialog who informs the user that the macro was completed successfully
        window = .JWindow~new()
        jpanel1 = .JPanel~new()
        jpanel2 = .JPanel~new()
        jlabel = .JLabel~new("Das Dokument mit der Bezeichnung '".file -
        "' wurde erfolgreich erstellt")
        jbutton     = .JButton~new("OK");
        bevelborder = .BevelBorder~new(0)

        jpanel1~~add(jlabel)
        jpanel2~~add(jbutton)

        window~getContentPane()~setLayout(.GridLayout~new(2,1));
        window~getContentPane()~setBorder(bevelborder);
        window~getContentPane()~~add(jpanel1)~~add(jpanel2)
        window~~pack()~~setLocationRelativeTo(window)~~setVisible(.true)-
        ~~setAlwaysOnTop(.true)

        jbutton~bsf.addEventListener('action', '', 'call next')
        Do Forever
                event = bsf("pollEventText")
                interpret event
        End



        next:
        window~dispose()
        Exit

        Exit


/***************************ROUTINE main_old*********************************/


::Routine main_old  -- Routine responsible for loading old documents

        oDesktop         = UNO.createDesktop() -- Create an UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader
        -- Get the componentloader interface from the desktop object
```

```
-- Open the content file of an old document in hidden mode
url              = makeUrl("C:\_old\".filename)
-- Get the document from the specified folder
props            = bsf.createArray(.UNO~propertyValue,1)
props[1]         = .UNO~PropertyValue~new
props[1]~Name    = "Hidden"
props[1]~Value   = box("boolean", .true)
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

xWriterDocument  = xWriterComponent~XTextDocument
-- Get the main interface of the textdocument
xText            = xWriterDocument~getText()
-- Get the text of the document

xTextCursor      = xText~createTextCursor()  -- Create a TextCursor
xWordCursor      = xTextCursor~XWordCursor   -- Create a WordCursor
xParagraphCursor = xTextCursor~XParagraphCursor -- Create a ParagraphCursor

xParagraphCursor~gotoendofParagraph(.true)
.local~file = xParagraphCursor~getString()


-- Loop that save each paragraph of the document to a variable
a = .false
i = 1
Do While a = .false
        xParagraphCursor~gotonextParagraph(.false)
        a = xWordCursor~isEndofWord()
        xParagraphCursor~gotoendofParagraph(.true)
        text.i = xTextCursor~getString()
        i=i+1
End

xWriterComponent~dispose()  -- Close the document



-- Open the template file which is specified in the content file

oDesktop         = UNO.createDesktop() -- Create an UNO Desktop service object
xComponentLoader = oDesktop~XDesktop~XComponentLoader
-- Get the componentloader interface from the desktop object

-- Open a template file
url = makeUrl("C:\_templates\".file) -- Get the textdocument from the specified folder
props = bsf.createArray(.UNO~propertyValue,1)
props[1] = .UNO~PropertyValue~new
props[1]~Name  = "Hidden"
props[1]~Value = box("boolean", .true)
xWriterComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0, props)

xWriterDocument  = xWriterComponent~XTextDocument /* Get the main interface
                                                     of the textdocument */
xText            = xWriterDocument~getText() -- Get the text of the document

xModel           = xWriterDocument~XModel -- Get the model from the textdocument
xController      = xModel~getCurrentController -- Get the controller for the model

/* With the help of the controller and XTextViewCursorSupplier
   we get the TextViewCursor */
xViewCursorSupplier = xController~XTextViewCursorSupplier
xViewCursor         = XViewCursorSupplier~getViewCursor

/* The XSearchable Interface gives the textdocument the
   possibility to search the content */
xSearchabel         = xWriterDocument~xSearchable
xSearchDescriptor   = xSearchabel~createSearchDescriptor
xSearchDescriptor~setSearchString("insert")
i=1
xFound   = xSearchabel~findFirst(xSearchDescriptor)
xFound.i = xFound

/* Loop which is searching for the searchstring within
   the document and saving the matchings to variables */
Do While xfound <> .nil
        .local~i=i
        i=i+1
        xFound = xSearchabel~findNext(xFound, xSearchDescriptor)
        xFound.i = xFound
End
```

```
        i=i-1
        -- Loop which gets the content that is positioned after the searchstring
        Do While i > 0
                xmodel      = xFound.i~XTextRange~getText() /* Get the textrange
                                                               of each searchstring */
                xposition   = xFound.i~XTextRange~getStart()
                xTextCursor = xmodel~createTextCursorByRange(xposition)
                xViewCursor~gotoRange(xTextCursor,.false)
                xWordCursor = xTextCursor~XWordCursor()

                xWordCursor~gotoendofWord(.true)
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                width.i=xWordCursor~getString()  -- Get the width of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                height.i=xWordCursor~getString() -- Get the height of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                text2.i=xWordCursor~getString()  -- Get the text of the textbox
                xWordCursor~gotonextWord(.false)
                xWordCursor~gotoendofWord(.true)
                f_height.i=xWordCursor~getString() -- Get the font height of the textbox
                xWordCursor~gotonextWord(.false)
                xSentenceCursor = xWordCursor~XSentenceCursor()
                xSentenceCursor~gotoendofSentence(.true)
                font.i=xSentenceCursor~getString() -- Get the font of the textbox

                f.i=xViewCursor~getPosition() -- Get the coordinates of the searchstring
                i=i-1
        End


        xWriterComponent~dispose()  -- Close the document



        -- Open a new document to insert forms and content

        xContext         = UNO.connect()
        -- Get a connection to the server and retrieve the XContext object
        xMCF             = xContext~getServiceManager -- Get the XMultiComponentFactory
        oDesktop         = UNO.createDesktop() -- Create an UNO Desktop service object
        xComponentLoader = oDesktop~XDesktop~XComponentLoader
        -- Get the componentLoader interface from the desktop object

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

        xWriterDocument  = xWriterComponent~XTextDocument
        xText            = xWriterDocument~getText()
        xServiceManager = xWriterDocument~XMultiServiceFactory
        -- Get the service manager of the textdocument

        /* Save objects that are often used in the program as '.local' variables.
           With this they can be accessed like global variables in the program.  */
        .local~Context         = xContext
        .local~MCF             = xMCF
        .local~ServiceManager  = xServiceManager
        .local~WriterComponent = xWriterComponent
        .local~Text            = xText


        i = .i
        /* Loop that creates textboxes at the position of the searchstrings
           with the specified properties out of the content of the template.*/
        Do While i>0
                textbox.i = create_form("TextField", width.i, height.i, f.i)
                -- Create a textbox with the routine 'create_form'
                textbox.i~setPropertyValue("Text", text.i) -- Assign text to the textbox
                textbox.i~setPropertyValue("MultiLine", box("boolean", .true))
                -- Set the textbox multiline
                FontDescriptor        = .bsf~new("com.sun.star.awt.FontDescriptor")
                -- Create a fontdescriptor
                FontDescriptor~Height = f_height.i
                FontDescriptor~Name   = font.i
                textbox.i~setPropertyValue("FontDescriptor", FontDescriptor)
                -- Assign the fontdescriptor to control the font
                i=i-1
        End
```

```
        xTextCursor       = xText~createTextCursor()
        -- Create a TextCursor within the document
        .local~TextCursor = xTextCursor
        xTextCursor~XDocumentInsertable~insertDocumentFromURL(-
        "file:///C:/_templates/".file,.UNO~noProps)
        xModel            = xWriterDocument~XModel -- Get model from the textcomponent
        xController       = xModel~getCurrentController()
        -- Get the controller of the textdocument

        -- Get the XViewCursor with the help of the controller and the XTextViewCursorSupplier
        xViewCursorSupplier = xController~XTextViewCursorSupplier
        xViewCursor         = XViewCursorSupplier~getViewCursor()
        .local~ViewCursor   = xViewCursor

        i=.i
        -- Loop which replaces all keywords of the template used for inserting textboxes
        Do While i > 0
                xReplaceable       = xWriterDocument~XReplaceable
                xReplaceDescriptor = xReplaceable~createReplaceDescriptor()
                -- Create the replacedescriptor with the XReplaceable interface
                xReplaceDescriptor~~setSearchString("insert "width.i" "height.i -
                " "text2.i" "f_height.i" "font.i)~~setReplaceString("")
                -- Set search and replace string
                xReplaceable~replaceAll(xReplaceDescriptor)
                -- Replace all strings matching the searchstring
                i=i-1
        End


        Call textsection         -- Insert a protected textsection

        i=.i
        -- Loop which changes properties of the textboxes
        Do While i > 0
                textbox.i~setPropertyValue("Border", box("short",0))
                -- Set the border invisible
                textbox.i~setPropertyValue("ReadOnly", box("boolean",.true))
                -- Set the content ReadOnly
                i=i-1
        End

        Call switch_design_mode  -- Switch the design mode from design to live mode
Exit


/***************************ROUTINE state_window*****************************/

::Routine state_window  -- Routine that opens a jwindow which is pausing the macro


        -- create java swing objects
        label    = .JLabel~new("")
        label2   = .JLabel~new("MAKRO ANGEHALTEN",0)
        label3   = .JLabel~new("-----------------------------",0)
        label4   = .JLabel~new("Bitte Inhalt eingeben!",0)
        label5   = .JLabel~new("Um Makro fortzusetzen,",0)
        label6   = .JLabel~new("bitte auf FORTSETZEN klicken!",0)
        window   = .JWindow~new();
        next     = .JButton~new("FORTSETZEN");
        cancel = .JButton~new("ABBRECHEN");
        icon     = .Icon~new("ma.png");
        jbuttonp = .JPanel~new();
        panel1   = .JPanel~new();
        panel2   = .JPanel~new();
        etchedborder = .EtchedBorder~new(0)

        -- set the layout for the jpanels and add objects
        jbuttonp~setLayout(.GridLayout~new(2,1))
        jbuttonp~~add(next)~~add(cancel);
        panel1~setLayout(.GridLayout~new(5,1))
        panel1~~add(label2)~~add(label3)~~add(label4)~~add(label5)~~add(label6)
        panel2~~add(label);


        -- Set the jwindow to use the GridBagLayout and add objects
        window~getContentPane()~setLayout(.GridBagLayout~new());
        window~getContentPane()~setBorder(etchedborder);
        window~getContentPane()~~add(panel1)~~add(jbuttonp)~~add(panel2);

        -- Set the size of the jwindow
        toolkit=.Toolkit~getDefaultToolkit()
        screenSize = Toolkit~getScreenSize()
```

```
        boxw = screenSize~width()/3
        boxh = screenSize~height()/7
        Parse var boxw boxw '.'
        Parse var boxh boxh '.'

        If boxw<426 Then boxw=426
        If boxh<146 Then boxh=146


        -- Get the Size of the screen without the task bar
        ge = .GraphicsEnvironment~getLocalGraphicsEnvironment()
        maximumWindowBounds = ge~getMaximumWindowBounds();
        resx= maximumWindowBounds~width()
        resy= maximumWindowBounds~height()

        -- Get the position of the jwindow in the right corner of  the screen
        boxpx = resx - boxw
        boxp = resy - boxh

        -- Set the location, size and visibility of the jwindow
        window~~pack()~~setLocation(boxpx,boxp)~~setSize(boxw,boxh)-
        ~~setVisible(.true)~~setAlwaysOnTop(.true)

        label~setIcon(icon);

        -- Add eventhandling to the jbutton
        next~bsf.addEventListener('action', '', 'Call next')
        cancel~bsf.addEventListener('action', '', ".local~c= true")
        cancel~bsf.addEventListener('action', '', 'Call cancel')


        /* A never ending loop which execute the messages from the
           eventhandler as a rexx program */
        Do Forever
                event = bsf("pollEventText")
                interpret event
        End

        Exit

        next:
        window~dispose()
        Exit

        cancel:
        window~dispose()
        .WriterComponent~dispose()
        Call create_gui
        Exit


/*************************ROUTINE switch_design_mode**************************/


::Routine switch_design_mode  -- Routine to switch the design mode from design to live mode
        url                    = bsf.loadClass("com.sun.star.util.URL") -- Load the class URL
        aToggleURL             = bsf.createArray(url, 1) -- Create an array to store the URL object
        aToggleURL[1]          = .bsf~new("com.sun.star.util.URL")
        aToggleURL[1]~Complete = ".uno:SwitchControlDesignMode"

        xmodel = .WriterComponent~XModel -- Get the current model of the document
        xController = xmodel~getCurrentController  -- Get the XController interface
        xDispatchProvider = xController~getFrame~XDispatchProvider -- Get the dispatch provider

        -- Use an URLTransformer to parse the url
        frameDesktop = .MCF~createInstanceWithContext("com.sun.star.util.URLTransformer",.Context)
        xURLTransformer = frameDesktop~XURLTransformer -- Initialisation of the XURLTransformer interface
        xURLTransformer~parseStrict(aToggleURL)        -- Parse the url
        xDispatcher = xDispatchProvider~queryDispatch( aToggleURL[1],"", 0 )
        -- Get all dispatcher from the specified url
        xDispatcher~dispatch( aToggleURL[1], .UNO~noProps )
        -- Execute a dispatcher from the given url


/*************************ROUTINE textsection*******************************/


::Routine textsection  -- Routine to insert a protected textsection accross the whole document
        .ViewCursor~gotoStart(.false)
        .TextCursor~gotoEnd(.true)      -- Mark the whole document as a TextRange
        xTextSection = .ServiceManager~createInstance("com.sun.star.text.TextSection")
        -- Create an instance of the textsection
```

```
        xTextSection~XNamed~setName("xTextSection")  -- Set a name for the textsection
        xTextSectionprops = xTextSection~XPropertySet()
        xTextSectionprops~setPropertyValue("IsProtected",box("boolean",.true))
        -- Set the property value IsProtected
        TextSection = xTextSection~XTextContent()
        .Text~insertTextContent(.TextCursor,TextSection,.true) -- Insert the textsection


/*************************ROUTINE create_form*******************************/


::Routine create_form -- Routine to create and initialize a shape for the document
        xMSF = .WriterComponent~XMultiServiceFactory
        xControlShape = xMSF~createInstance("com.sun.star.drawing.ControlShape")~XControlShape
        -- Create a control shape

        FormComponent = "com.sun.star.form.component."ARG(1)  -- Create a form component
        xControlModel = .MCF~createInstanceWithContext(FormComponent,.Context)~XControlModel
        -- Create a control model

        -- Set the position and size of the form
        xControlShape~setSize(.bsf~new("com.sun.star.awt.Size", ARG(2)*100, ARG(3)*100))
        -- The size is specified in 100th/mm
        xControlShape~setPosition(ARG(4)) -- Set the position specified in 100th/mm
        xPropertySet = xControlShape~XPropertySet  -- Get the propertyset of the shape
        xPropertySet~setPropertyValue("AnchorType", bsf.getConstant(-
        "com.sun.star.text.TextContentAnchorType", "AT_PARAGRAPH"))
        -- Adjust the anchor to the paragraph

        xControlShape~setControl(xControlModel) -- Set the ControlModel of the shape

        -- Add the shape to the shapes of the document
        xDrawPageSupplier = .WriterComponent~XDrawPageSupplier
        xDrawPage         = xDrawPageSupplier~getDrawPage()
        xShapes           = xDrawPage~XShapes
        xShapes~add(xControlShape)

        return xControlModel~XPropertySet /* Returns the XPropertySet interface
                                             to give the possibility of adjustments */
```

Figure 49: Sourcecode of the Template Handling Tool