Institute for
Management Information Systems
Vienna University of Economics and Business Administration

Institut für Betriebswirtschaftslehre und Wirtschaftsinformatik, Wirtschaftsuniversität Wien
UZA II 2.Ebene, Augasse 2-6, A-1090 Wien, Austria

WIRTSCHAFTSUNIVERSITÄT WIEN
BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:
Automatisierung von Open Office – ooRexx Beispiele

Englischer Titel der Bakkalaureatsarbeit:
Automating Open Office – ooRexx Nutshells

Verfasserin/Verfasser: **Josef Frysak**

Matrikel-Nr.: **0052398**

Studienrichtung: **Wirtschaftsinformatik**

Kurs: **Electronic Commerce – Vertiefungskurs 6**

Textsprache: **Englisch - English**

Betreuerin/Betreuer: **Prof. Dr. Rony G. Flatscher**

Semester: **Sommersemester 2008**

Ich versichere:

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

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

dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

	Datum		Unterschrift	

Abstract

This thesis is about automating Open Office using ooRexx. The theoretical part contains informations about the UNO structure, which enables Open Office to be remote controlled by another program. Furthermore, the paper shows how to start a macro inside as well as outside of Open Office and how to prepare examples capable of being run inside and outside of Open Office. This work also describes, how to call and address a macro out of another macro or a tool bar. This also includes referencing ooRexx macros, which have been installed as part of an addon. It also contains various helpful macros (nutshells) regarding Writer documents, Calc documents, and basic Open Office operations. Importing source code using Vim editor and creating an ooRexx addon out of a Library, containing ooRexx macros, are only two examples.

Thanks

At this point I want to send a lot of thanks to Prof. Dr. Rony G. Flatscher for his great help during the creation of this paper. His many thought-provoking impulses have been of great value to me. He also had a lot of work with his BSF4Rexx Library, in order to get my examples to run.

Table of Contents

1 Introduction

This Bachelor thesis shall give the reader an overview on how to automate OpenOffice.org, especially concentrating on Open Office Writer and Calc.

1.1 Initial Words

In this paper automation has the meaning of controlling, i.e.: sending commands to, the Open Office Suite from an external program or directly from within Open Office using a macro[1]. To provide examples of such programing snippets the author used ooRexx. Furthermore many examples within this paper have been taken from Andrew Pitonyaks article "Useful Macro Information For OpenOffice"[2] and translated to ooRexx scripts[3]. Additionally this paper has been enriched by other examples, created by the author. This paper does not serve as a reference book for Open Office API or ooRexx, but it tries to communicate the structure behind it and to demonstrate basic commands to access the Open Office environment as well as the Open Office Writer and Calc program. Another goal of this writing is to create platform independent examples by using URL strings to store paths and by using ooRexx internal functions to convert these URLs to operating system dependent path information.

1.2 Used Environment and Versions

The examples mentioned before have been tested under the following conditions:

- Windows XP Service Pack 2,
- OpenOffice 2.4.0, 2.4.1,
- Java 1.6.0_04,
- ooRexx 3.2.0 build of 30.10.2007[4],
- BSF4Rexx 2.6 („Vienna Version") of date 03.09.2008.

---

1 A macro is a sequence of commands, understandable by a specific program.
2 This paper is downloadable at http://www.pitonyak.org/AndrewMacro.odt. Cf. [UMIFOO].
3 A script is a piece of code, which can be executed by an interpreter and is readable by humans.
4 Go to command line shell and type „rexx -v“ to determine the version of the ooRexx interpreter.

1.3 Andrew Pitonyaks Nutshells

As mentioned in the introduction many examples provided in this paper have been taken from Andrew Pitonyaks paper "Useful Macro Information For OpenOffice". Cf. [UMIFOO]. This paper describes how to write advanced macros in Open Office. These macros consist of a language derived from Visual Basic, and can be executed inside an Open Office program.

1.4 ooRexx & BSF4Rexx

The following two chapters describe, why the programming language ooRexx has been chosen, and why ooRexx gets in need of the BSF4Rexx library in order to execute tasks in Open Office.

1.4.1 ooRexx

The programing language, the examples are written in, is ooRexx. This language has been chosen for many purposes.

- Firstly, it is distributed under CPL[5], which means easy and charge free access to this software for the author and the reader alike. Cf. [ooRexx01].
- Secondly, it is easy to learn this language. By using a limited amount of basic commands, one is able to learn this language quite fast and if there is a need for increased functionality, additional libraries can be loaded. Cf. [ooRexx01].
- Thirdly, ooRexx provides an easy to use command line execution. If some command within a script cannot be found by the interpreter, it is executed like a shell command. This shell execution can also be called explicitly. The shell execution will be helpful to start and deactivate an Open Office acceptor and to call other external programs like Vim. Cf. [ooRexx01].
- Another advantage is the object orientation of ooRexx, which allows to use classes and objects, and to reuse or inherit existing code. Cf. [ooRexx01].
- Furthermore ooRexx does not need to compile the scripts before executing them. These simple text files can be executed exactly as they are. Cf. [ooRexx01].

[5] CPL is an acronym for „Common Public License“.

Finally you may also find many examples for automating Open Office with ooRexx directly at the OpenOffice.org developer homepage.

1.4.2 BSF4Rexx & BSF.CLS

The first challenge is to establish a connection to Open Office. Notice that we actually want to connect to the UNO environment which will be discussed in the next chapter 1.5. The basic installation of ooRexx is not able to establish a connection, creating UNO objects, or sending commands without additional functions. Therefore the BSF4Rexx library, which allows us to use the Java Virtual Machine, must be installed to fulfill our needs. The Java Bean Scripting Framework for ooRexx allows to use Java classes and objects in ooRexx scripts. It also enables Java to execute these external programs and scripts of ooRexx. Furthermore it takes care of type conversion, because ooRexx is using untyped variables. Additionally, it provides special functions to create, access and manipulate Java arrays. Cf. [Jakarta01].

1.5 The UNO Environment

To remote control the open office package, a remote procedure call environment[6] is needed in order to send commands. The Universal Network Object (UNO) provides such a RPC system, and is not only able to send single commands, but to share one and the same programming object between two independent programs. If there is any change made to the object, this change is also done to the object held by the other program.

1.5.1 Architecture

To document objects UNO uses the UNOIDL[7], a CORBA like protocol able to transport a whole object trough a network. Cf. [ProfUNO01].

Another important part is the UNO Remote Protocol, short URP, used to transport the commands and references to objects. It consists of a header and a payload. The payload contains the messages or results to send. To accelerate this protocol UNO uses two caches[8]. Cf. [URP01].

---

6 An RPC environment is exporting functions of a program to allow another program to remote control its functionality.

7 UNOIDL is an acronym for „UNO Interface definition language“.

8 One cache is storing the last object used, the other one is an indexed list of UNO-Objects.

Language Bindings

Because UNO is independent of programming languages, language bridges are needed to transform UNO Objects to programming language specific objects. Cf. [ProfUNO01]. This fact is denoted in figure 1 below.

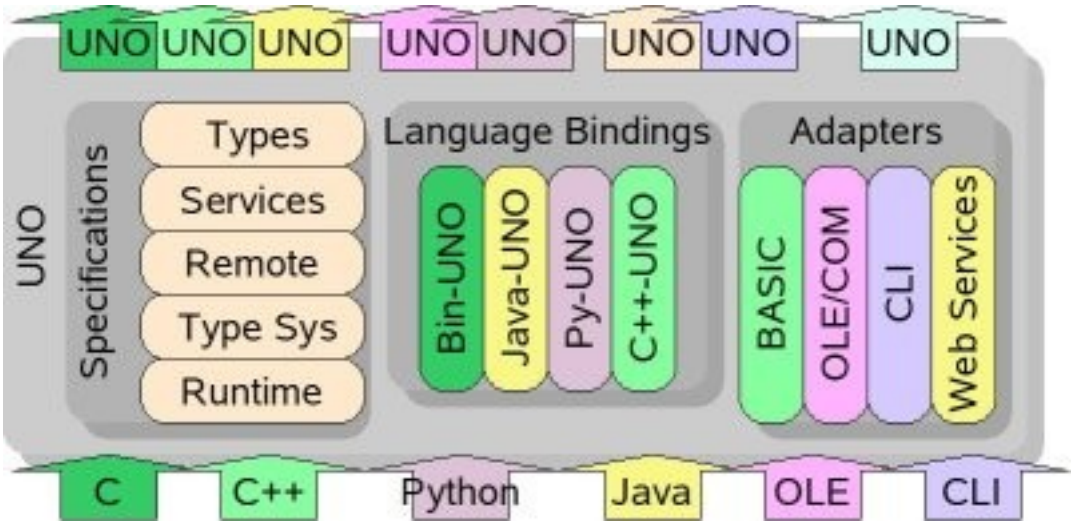

Figure 1: The UNO language bindings.

Cf. [http://images.google.at/imgres?imgurl=http://wiki.services.openoffice.org/w/images/b/bb/Uno-Arc.jpg&imgrefurl=http://wiki.services.openoffice.org/wiki/Uno&h=193&w=400&sz=27&hl=de&start=2&um=1&tbnid=DHpFZvSC0bRtKM:&tbnh=60&tbnw=124&prev=/images%3Fq%3Duno-arc.jpg%26um%3D1%26hl%3Dde%26sa%3DN, retrieved on 21.07.2008].

Services, Interfaces and Factories

UNO consist of two main structures. The service, which states an object, holding all the methods and properties needed to interact with the peer program. To use a service, one needs to query an interface, provided by the service, containing the properties or methods he wants access to. A service not only provides several interfaces, it is also integrating other services as a part of it. Some services are also capable of creating UNO objects, and therefore they are also able to create other services. UNO components with such an ability are called factories, or service factories. Cf. [ProfUNO02].

1.5.2 Implementation (the UNO-URI)

To send the commands and data from one program to another, the UNO environment uses a TCP/IP based network connection. Therefore, Open Office has to be put into listen mode, before one of the examples can be started. To do so, an UNO-URI must be defined. It is containing all the information to set up a server, which is able to accept UNO connection attempts. Figure 2 below displays the structure of such a string.

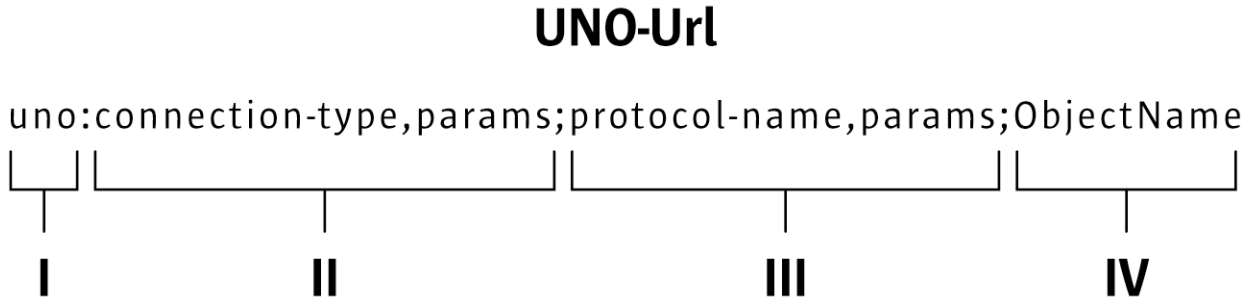

Figure 2: The UNO-URL-String

Cf.[http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/uno-url.png, retrieved on 03.04.2008].

I) The first string identifies the following URI string. To identify an UNO-URL the string „`uno`:" is being used. Cf. [ProfUNO03].

II) The next entry is the connection type we want to use. Usually a socket connection is considered to be most suitable. Normally such a socket connection also needs additional information like the host IP address and the port number to listen at. This additional information can be specified by using a comma separator. In this paper we will always use „`socket,host=localhost,port=2002;`". The semicolon at the end of the string is just used to state the end of this part. Cf. [ProfUNO03].

III) The next information, tells the server which protocol is used to transport the commands and objects. As mentioned before, the protocol is URP, so in this paper we will always state „`urp;`" here. Here we also use a semicolon to define the end of this information. Cf. [ProfUNO03].

IV) The last information is just used by the client[9]. When our client programs try to connect to the UNO environment they need to state the name of a service object used to control Open Office, because the UNO environment is independent from Open Office. Cf. [ProfUNO03].

Here are two short examples, which can also be found in our nutshells:

- to start Acceptor or Server:
  `uno:socket,host=localhost,port=2002;urp;` Cf. [ProfUNO03]
- for Client connection:
  `uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager`
  Cf. [ProfUNO03]

Creating such a string is only necessary, if the macro shall be executed from outside Open Office. Also, BSF4Rexx provides a method which will start a connection with a predefined string. But these issues will be discussed in more detail at chapters 3 and 4.

[9] For example: The macros provided in chapter 6 state such clients.

2 Preparations for Trial of the Examples

Before a macro can be executed, some installations and configurations must be done:

- Open Office must be installed to remote control it.
- Java is actually responsible for the remote procedure calls.
- The ooRexx interpreter is used to execute the ooRexx scripts,
- and the BSF4Rexx library serves as a bridge between ooRexx and Java.
- Finally the Vim[10] editor is needed by the macro that is importing code into a writer document.

The following chapters describe which steps to take, in order to be able to execute the macros delivered in this paper.

2.1 Install Open Office and Java

Installing Open Office[11] is quite simple. Download Open Office from Public Open Office Homepage. There are two kinds of Open Office Packages. The first one installs Open office only, the other one also contains the latest Java version. If Java is already installed on the target machine download the single version, otherwise download the complete package. After the download, the installation can be started by executing the Open Office installer.

To make things easier it is recommended to do the standard installation, and install the software to the default path. This will help the reader to follow the instructions of this paper more easily.

To support external macro calls, add the "`juh.jar`", "`jurt.jar`", "`ridl.jar`", and "`unoil.jar`" to the Classpath environmental variable of Java. Cf. [ProfUNO04].

When installing BSF4Rexx the Java libraries will be added to the Classpath automatically. The libraries have been listed here to help Java programmers finding the libraries they need.

---

[10] Vim is an open source text editor with many features. Originally it was developed for the Linux community, but now it is also available for windows. It is downloadable at http://www.vim.org/download.php. Cf. [VIMONLINE].

[11] Open Office is downloadable at http://download.openoffice.org/other.html. Cf. [OOFFICE]. To download the package including Java just check the check box above the table.

Next prepare Open Office by starting one of its programs like Calc or Writer. There select Tools from the menu bar. Within tools list select options as seen in figure 3.

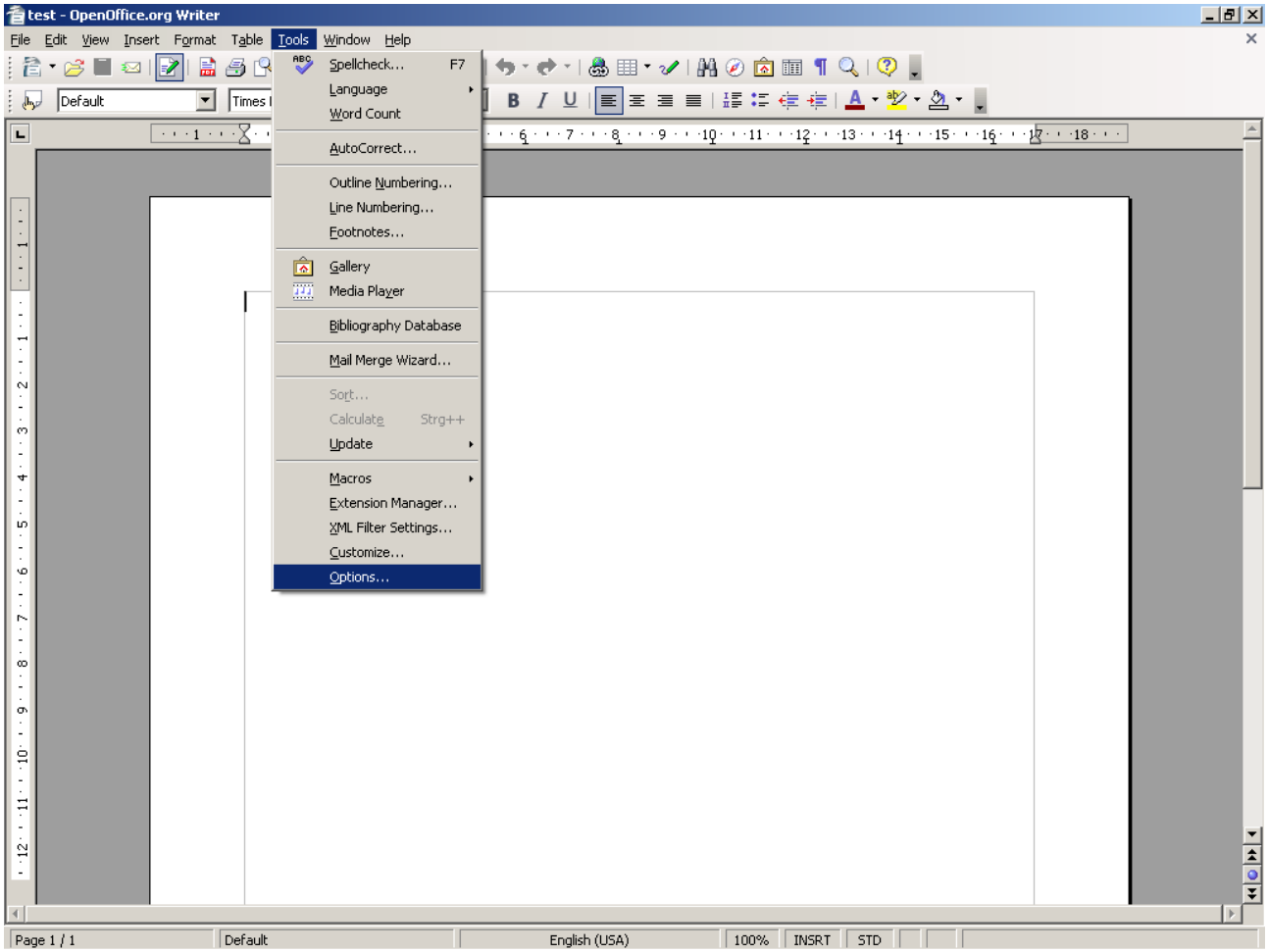

Figure 3: Open Office options.

Within the options menu select „`Memory`" and check „`load OpenOffice.org during system startup`" like in figure 4. This activates the quick starter being activated on system startup.

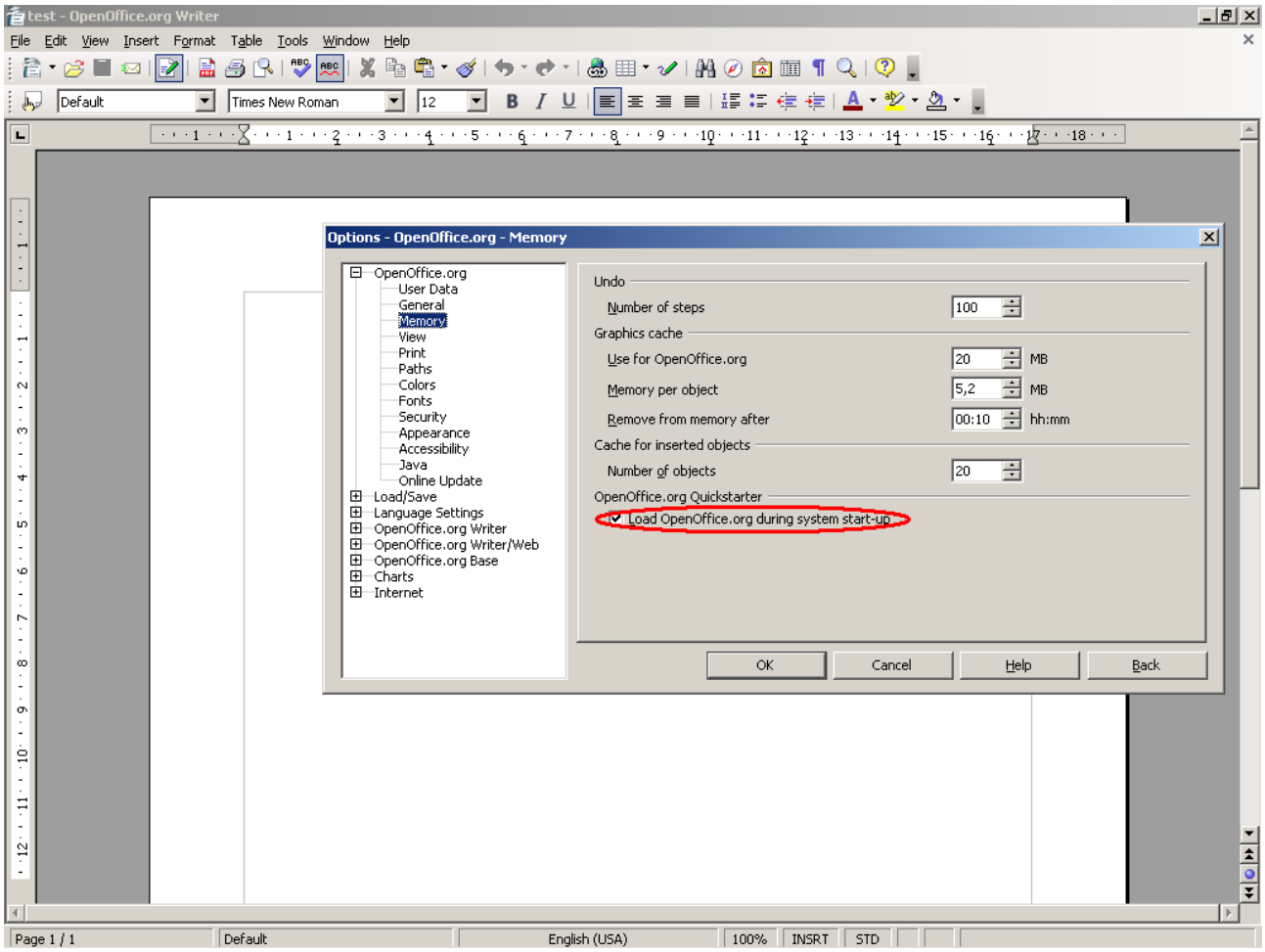

Figure 4: Quickstarter option.

This allows keeping a connection established, even if the document window is closed.

Then select „`Java`" and activate „`Use a Java runtime environment`" to allow Open Office to use Java and BSF4Rexx. Cf. [OOoInstall01]. Figure 5 shows the corresponding options screen.

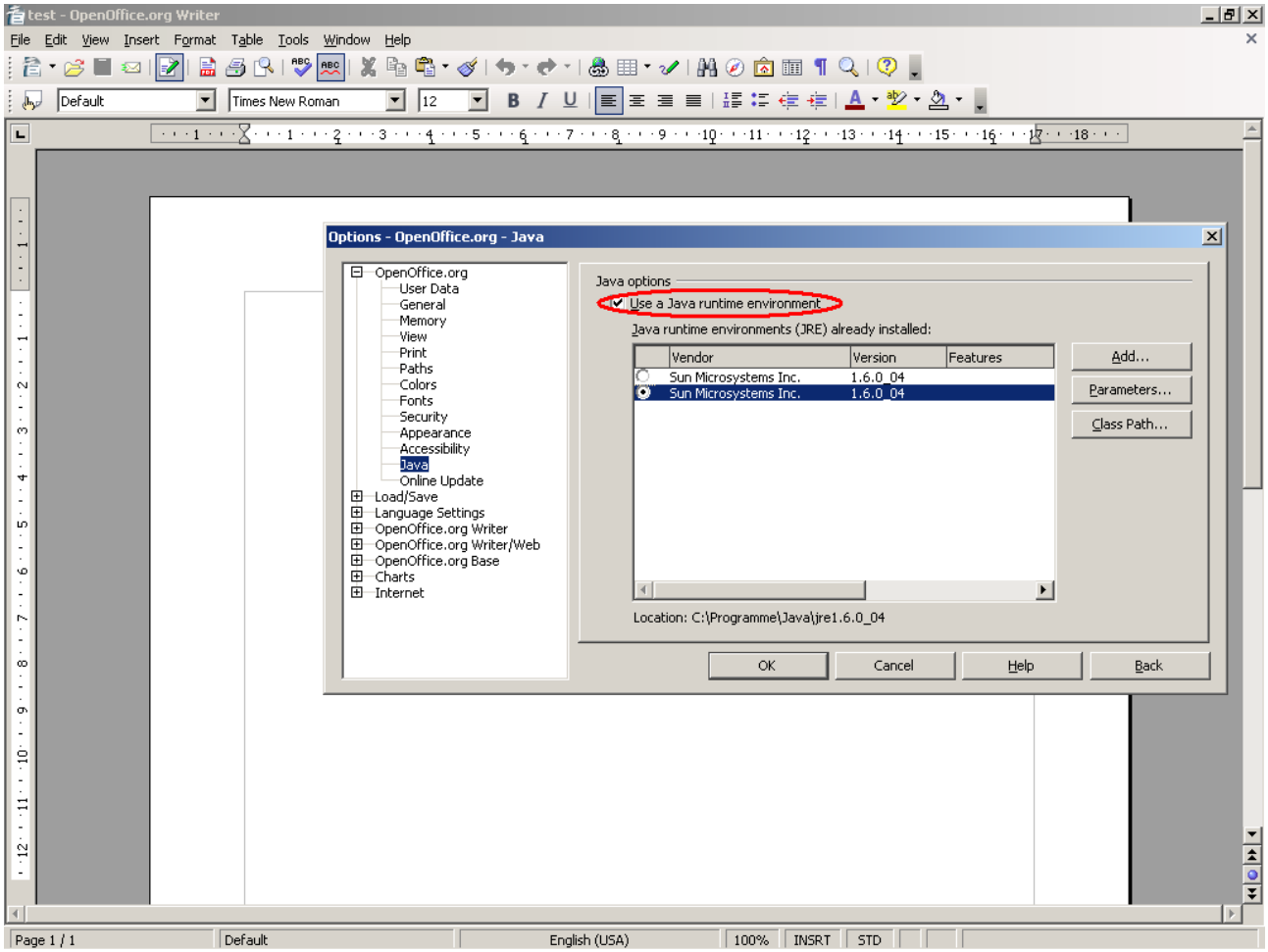

Figure 5: Java runtime environment options.

This is required to let Open Office use Java.

2.2Install ooRexx

Download the latest ooRexx version[12] and install it by executing the downloaded installer.

When installing you do not have to install to the default path, but using the default path is easier and will help understanding the explanations of this paper.

[12] ooRexx is downloadable at http://www.oorexx.org/download.html. Cf. [GETREXX].

2.3 Install BSF4Rexx

Before starting the installation of BSF4Rexx[13], first all currently running instances of Open Office must be shut down. The Quickstarter must be closed too, if active.

Now download the latest version. The download contains an archive, which has to be extracted to a folder. Next a command line tool is used to navigate to the folder where BSF4Rexx has been extracted to. There, "`setupBSF.rex`" is called[14] which will do the setup of BSF4Rexx and will create several additional files. One of these additional files is called "`installBSF.cmd`"[15] using Windows or "`installBSF.sh`" using Linux. It will set the system path environment variable to contain the path to BSF4Rexx and the Java Classpath environmental variable which allows Java to find the BSF4Rexx engine. Cf. [BSFInstall01].

Then "`setupOOo.rex`" must be called. This will prepare the BSF4Rexx installation for Open Office and it will create additional files too. If this script displays an error, because it was not able to find the current Open Office installation, the script has to be called again supplying the absolute path to the Open Office installation as parameter.

One of the additional files created is named "`installOOo`". Using Windows it is a "`.cmd`" file on Linux a "`.sh`" file will be present. This file needs to be executed in order to install the Open Office ooRexx support. Cf. [OOoInstall01].

2.4 Install Vim

One example provided in this paper uses the open source editor "`Vim`", especially its graphical user interface "`gvim`". To provide this program to the script, the Vim installer needs to be downloaded and installed.

Because the script executes "`gvim`", the path where "`gvim`" is located must be added to the systems path environment variable, to enable ooRexx to find the executable file.

---

[13] BSF4Rexx is downloadable at http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/. Cf. [GETBSF].

[14] ooRexx scripts are executed by calling the interpreter "`rexx`" submitting the invoked script as its first parameter. If BSF4Rexx is installed and the script intends to use Java, then use the "`rexxj`" command to call the interpreter, submitting the invoked script the same way as described before.

[15] To execute shell scripts under windows, just enter the name of the script into the command line and press enter.

3 The Open Office Acceptor (UNO – Server)

When writing macros for Open Office with ooRexx, it is possible to start these macros outside of Open Office. As mentioned before, the connection between the two programs, i.e.: the macro and Open Office, is a network connection with a server and a client role. Therefore it is required to start the Open Office acceptor, an UNO network server, to allow the macro to connect to Open Office.

There are 3 ways to start the UNO Environment:

- Configuring Open Office to load the server on startup,
- starting the server using the command line,
- and starting the server from within an ooRexx script.

The First two ways are done manually, while the last way is some sort of automatic starting from within our program. In any case keep in mind, that in order to keep the acceptor or server alive, even if the user closes the Open Offices applications, the Quickstarter menu must be enabled. Independent of the way selected, the acceptor definition string is needed to state the attributes of the server.

3.1 Way 1: Configuring startup options

One way to start the UNO listening mode of Open Office is to change its settings at "`Setup.xcu`". This file contains many configuration details which are being used every time Open Office is started. Also, the configuration is written in XML style, so the file can be opened and changed by a simple text editor.

To find the containing folder of this Setup file the installation path of Open Office has to be located. Within the installation folder follow the "`<InstallPath>/share/registry/data/org/openoffice/`" path. There the "`Setup.xcu`" can be found. Cf. [ProfUNO03a].

After opening the file following tag must be searched for:

`<prop oor:name="ooSetupConnectionURL">`.

If this tag does not exist create it within the

`<node oor:name="Office">` tag.

Now change the created or found tag to:

```
<prop oor:name="ooSetupConnectionURL">...
<value>socket,host=localhost,port=2002;urp;StarOffice.ServiceManager</value>
</prop>
```

Cf. [ProfUNO03a].

Conclusion

This method is affecting the whole installation of Open Office. This means, every time Open Office or one of its programs is started, the server starts too, using the settings stated int the value tag. By enabling the quick start menu on start up, this server is always active.

3.2 Way 2: Using the Command Line

We may also start the server from the command line, by simply calling "`soffice.exe`" followed by some parameters.

But first the installation path of Open Office has to be located again. Within the installation path go to "`<Install Path>/program/`" path. There the "`soffice.exe`" executable file is located. Now we call:

```
soffice.exe -accept=socket,host=localhost,port=2002;urp;.
```

This will start the Open Office UNO acceptor. Cf. [ProfUNO03a].

If we want to keep the server running even if the user closes the application we might use following command to start the Quickstarter at the same time:

```
soffice.exe -accept=socket,host=localhost,port=2002;urp; -quickstart.
```

By typing:

```
soffice.exe -unaccept=socket,host=localhost,port=2002;urp;
```

the listening mode of Open Office is closed again.

Windows hint

If these commands shall be executed in a batch file or an ooRexx script, the "start" command must be added at the beginning of the string. Otherwise an error will occur.

Conclusion

When using this method, the server stops as soon as the Application is being terminated.

To avoid this, the Quickstarter must be enabled. But using this method one has the ability to close the connection at any time.

3.3 Way 3: Command line Execution in ooRexx

This method is very similar to the method explained before. But now the execution command is called from within the ooRexx script. Again the Open Office installation path must be retrieved to locate "`soffice.exe`". Finally the ooRexx script looks something like this:

```
'START <INSTALL PATH>\PROGRAM\SOFFICE.EXE -ACCEPT=SOCKET,HOST=LOCALHOST,PORT=3801;URP; -QUICKSTART'
```

Normally writing an ooRexx script, the term "ADRESS CMD" is not necessary, but executing the script using "`rexxj`" command would result in a error message. Therefore it is recommended to use this term.

The Windows command "`start`" is also used here to avoid getting an error message from open office at startup when calling this macro under Windows. Using Linux this term should be removed, but the accept parameter should be enclosed by quotation marks. See Sourcecode 1.

Another problem using this method is, that slower computers will take some time to load and execute the Open Office listening server. So if the connection to open office is established directly after this call, an error might occur, because the server might not be ready to answer the request. Therefore the script needs to wait a few seconds to give the server some time to load, before executing the rest of this script.

Here an example:

x_StartListenMode.rex[16]

```
-- using rexxj command use ADDRESS function otherwise you get an error

-- find out which operating system is present and set up the shell execution
/* Normally ooRexx is able to determine the operating system itself. But calling
   the macro with rexxj or inside Open Office disables this ability.*/
if .uno~path.separator=";" then
do
   -- Windows
   -- the start command is necessary when using windows, otherwise an error occurs.
   -- also the location of the soffice program differs
   add = 'start C:\Programme\"OpenOffice.org 2.4"\program\'
   ADDRESS CMD
end
```

[16] Meaning of the Prefixes:

- x means this script is containing code that carries out basic tasks like starting the Open Office server or to start a Macro,
- e means the script does tasks related to the environment of Open Office like changing the path settings,
- w means that the script executes tasks related to Writer documents,
- c means that the script relates to Calc document operations.

```
else
do
   -- Linux
   shell=value("SHELL",,"ENVIRONMENT") -- get type of shell
   shell=substr(shell, shell~lastpos("/")+1) -- get shell name
   ADDRESS VALUE shell           -- set shell as command shell
   add = ''
end

command = add || 'soffice "-accept=socket,host=localhost,port=2002;urp;" -quickstart'

-- execute command
command

-- by using SysSleep we wait 2 seconds
call SysSleep 2;

::requires UNO.CLS
```

Sourcecode 1: x_StartListenMode.rex

Example specific references: Cf. [ProfUNO03a].

4 Invoking or Executing a macro

The next chapters describe the different ways of executing an Open Office macro.

4.1 Externally by ooRexx

Before starting a macro outside of Open Office make sure that the Classpath of Java is set correctly as mentioned in chapter 2.1 to enable Java to communicate with Open Office. Additionally, the path to BSF4Rexx should be added to the system path environment variable, so when starting a ooRexx program it can easily be executed based on any path by calling "`rexxj`" command. An easier way to do all these settings is to call "`setEnvironment4BSF`" and "`setEnvironment4OOo`". These two command files, which are automatically generated when installing BSF4Rexx as described at 2.3, will do the same settings as described before.

Next the the server is being started by one method described in the chapter before. Finally the "`rexxj`" command is called submitting the name of the macro as first parameter.

A simple Example: `rexxj c_AutoCalc.rex`

4.2 Inside Open Office

To execute a macro inside Open Office we first need to copy the macro to a path, where Open Office stores its macros. Open Office provides two paths to store macros. The user specific path and a general path for all users. The user specific path for ooRexx scripts can be found at "`<UserApplicationDataDirectory>/OpenOffice.org2/user/Scripts/oorexx`" and the general path is located at "`<OpenOfficeInstallPath>/share/Scripts/oorexx`". Using the English version of Windows XP the "`<UserApplicationDataDirectory>`" would be "`C:\Documents and Settings\<UserName>\Application Data`". Cf. [UMIFOO01].

Every macro storage folder consists of libraries[17], wich serve as containers for the macros. To add a macro manually, first such a library foldermust be created within the storage folder, and then the macro must be copied into the library folder. To make the macro

[17] In the file system they are directories.

available by Open Office it also needs to be registered to a xml description file located within the library.

To avoid searching for these folders and to create or change such a description file manually, there is a much easier way to import a macro using the Open Office Macro Organizer. To start it, run any Open Office program like Writer. There select "`Tools`" from the menu. Within the tools dropdown menu move to "`Macros`" and then to "`Organize Macros`" and select "`ooRexx`". See also figure 6.

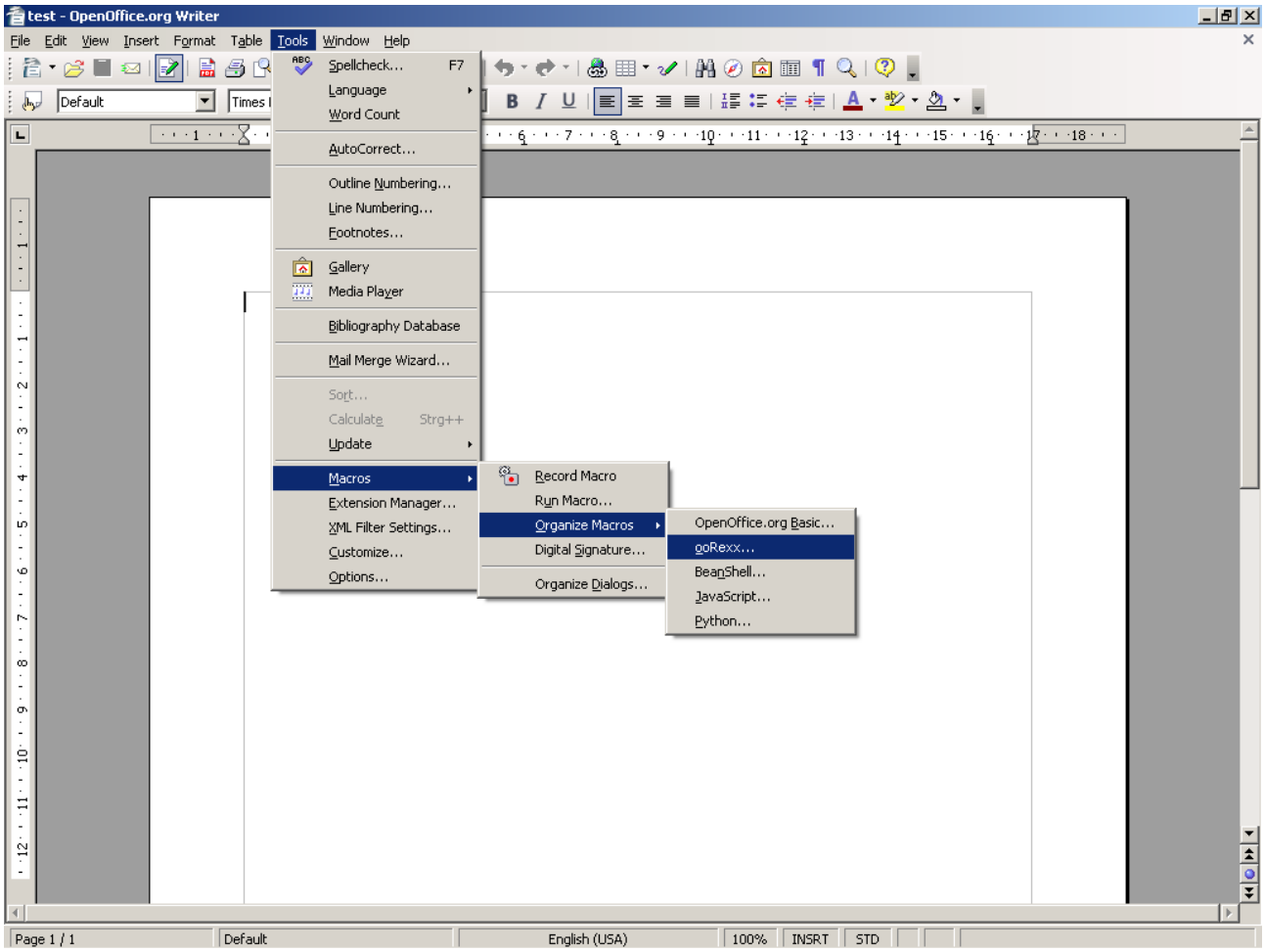

Figure 6: ooRexx macro organizer.

Now the organizer window appears and may look like in figure 7. There are three different macro storages, one can select:

- storage for user specific macros,
- storage of shared macros,
- macros stored in a document.

Opening "`My Macros`" will reveal the user specific ooRexx macro storage and the "`OpenOffice.org Macros`" folder is the link to the macros path that is shared by all users. If documents are open then these documents are displayed too, because every document is also able to store macros. But in this paper only the first two folders are relevant. By pressing the create button on the right, a new library is created, and by navigating into this library and pressing the same button again a new macro will be created. This action is also depicted in figure 7 below.

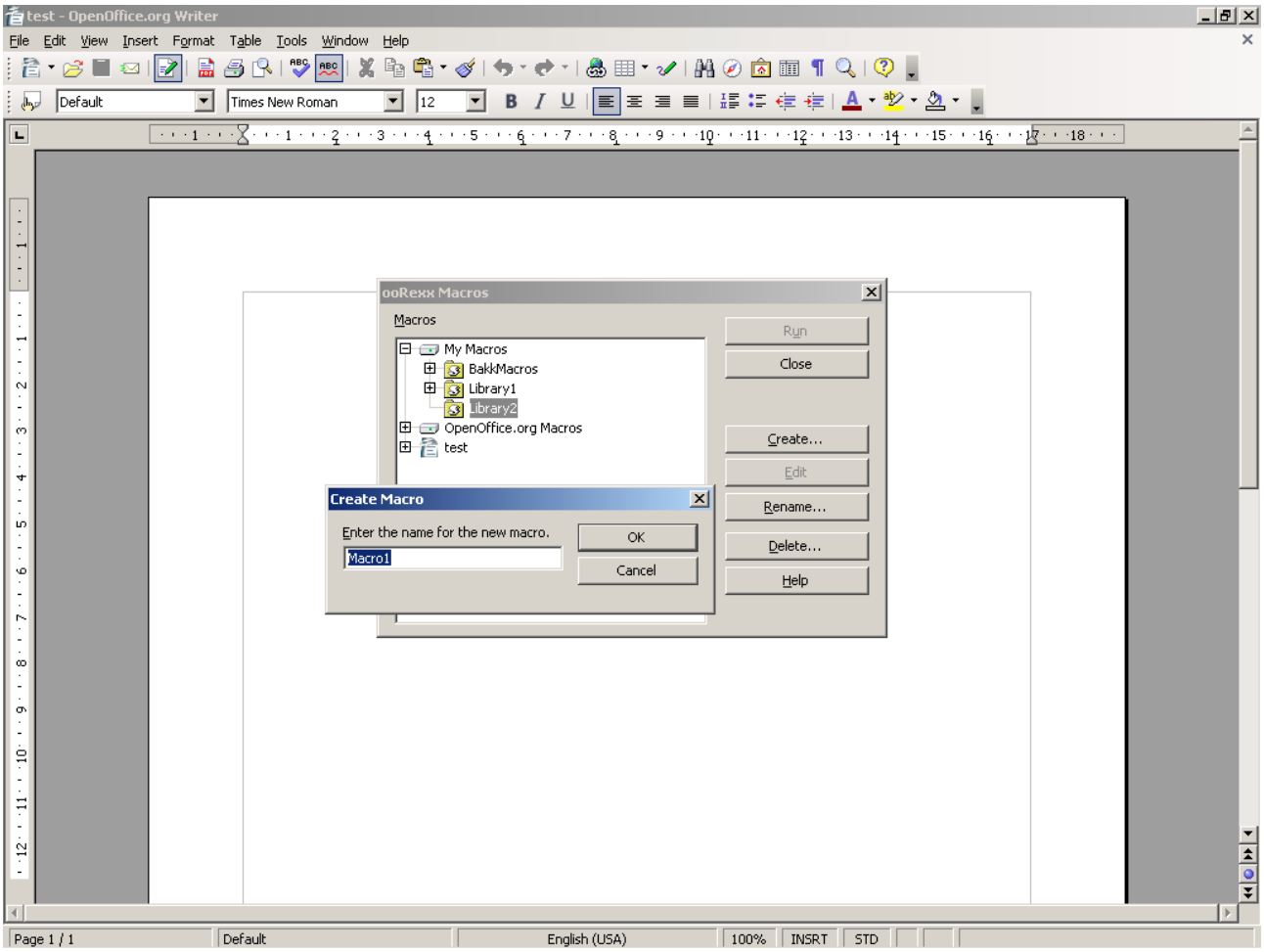

Figure 7: Create a new library and a new macro.

Selecting the macro and pressing the edit button will bring up a simple text editor of open office editing the previously selected macro. A newly generated script will contain a custom example, which can be deleted by pressing the "`Clear`" button at the bottom. Figure 8 shows a screenshot of the editor.

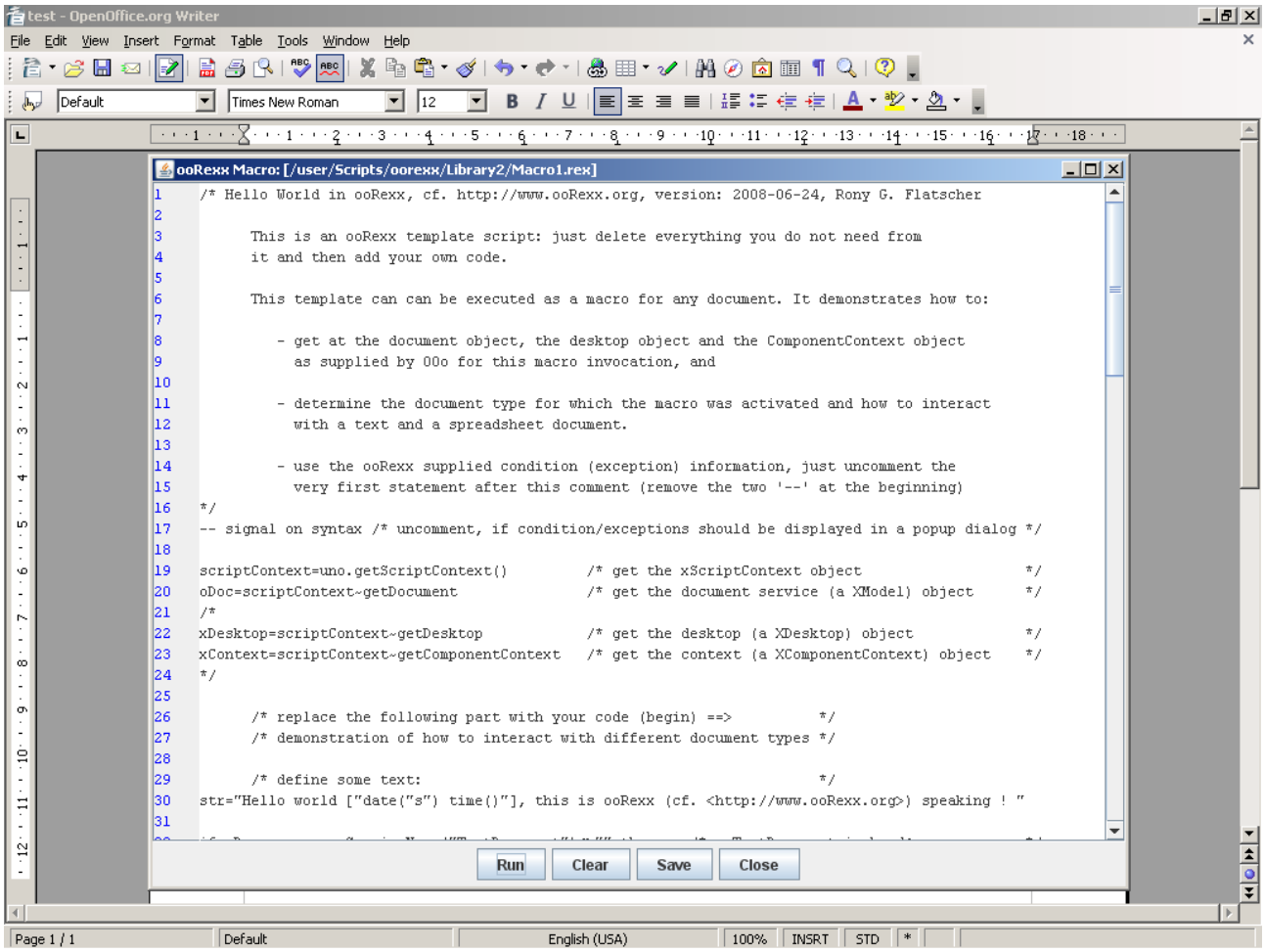

Figure 8: macro editor of open office (except star basic).

Next use some text editor to open the external script file and to copy the macros content to the Open Office macro editor. Pressing the "`Save`" button at the bottom of the editing window will result in the macro being stored.

This method is much easier than the one above, but it only allowes implementing one macro after another.

This text editor is also able to select the whole text at once (Strg+A), as well as doing copy (Strg+C) and paste (Strg+V) operations, by pressing a combination of keys.

If the macro has been successfully imported, the macro can be started by select "`Tools`" from the menu, navigating to "`Macros`" and select "`Run Macro`", as seen in figure 9.

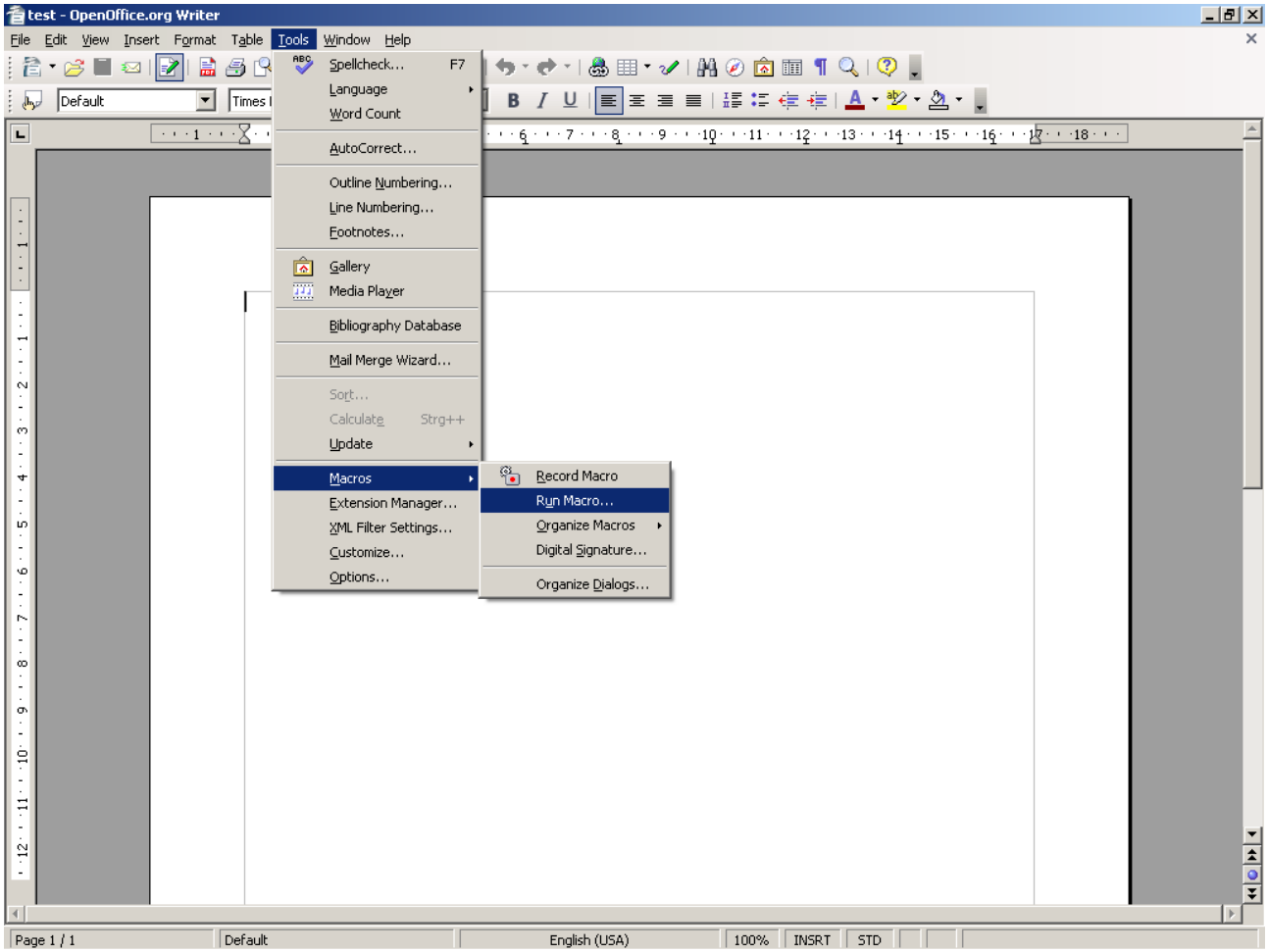

Figure 9: Run a macro.

Now the macro can be selected by navigating through the container and libraries to the macro that shall be executed, and executed by pressing the "`Run`" button on the right side as depicted in figure 10.

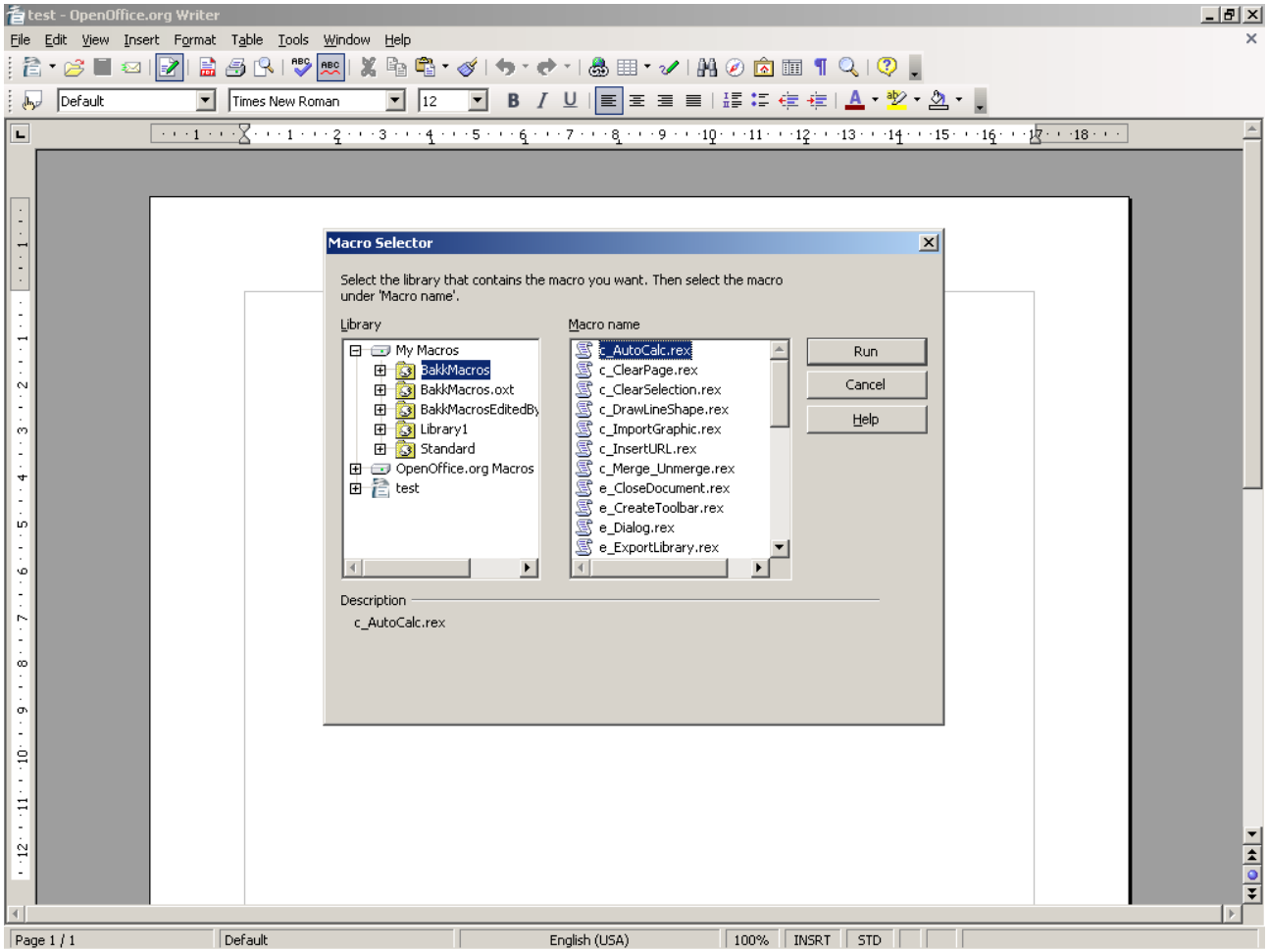

Figure 10: Select a macro to run.

Cf. [RunMacro01].

Also keep in mind, that macros might require a document of a specific type[18] in order to do its task. For example: A macro, that is doing a cell related task, cannot be used on a draw document.

4.3 Using Command Line

It is also possible to force Open Office to start a macro from the command line. To do so, the macro needs to be imported to Open Office, like explained above. If the macro has been implemented successfully, call the "`soffice`" executable and add a macro URL like this "`vnd.sun.star.script:<library>.<macroname>.rex?language=ooRexx&location=user`" as its only pa-

[18] A document of a specific type, for example, is a Writer- or Calc- document.

rameter. Open Office installations on Windows operating systems, contain the "soffice" executable within the Open Office installation path, in the subdirectory “program”. Using Linux the "soffice" executable is already added to the path environmental variable.

This example

```
soffice "vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=user"
```

will execute the "x_Sample.rex" macro located in the user specific macro container and the library named "BakkMacros".

If the library is located in the shared macro folder, the command should look like this:

```
soffice "vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=share"
```

To address a ooRexx macro within an Open Office Addon change the location string to "user" if the addon was installed at "My Macros", and append ":uno_packages/<filename>". The macro URL should then look something like this:

```
vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=user:uno_packages/BakkMacros.oxt
```

Cf. [UMIFOO02].

4.4 Out of a Macro or a Tool Bar

The definition of the location of the macro which should be called, when invoking a macro from within another macro, is quite the same as calling the macro from the command line. The only problem is to find out how to do the call within a macro using the Open Office API. The macro URL[19] identifies the macro that shall be executed. The way a tool bar button is told which macro to execute, in case of being pressed, also looks exactly the same.

The following example shows how to do such a call to another macro:

To execute another macro instantiate a "DispatchHelper" service and get its "XDispatchHelper" interface. To create a "DispatchHelper" service use the "XMultiServiceFactory" interface of the component contexts "ServiceManager"[20]. Also prepare a "XDispatchProvider" interface[21] of the the desktop service, to define which

[19] A description where the macro is located. It looks like this: vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=share. It is described in more detail in the chapter above.

[20] To create a new service call the "createInstance" function of the "XMultiServiceFactory" interface, which can be retrieved from the component contexts "ServiceManager". By calling this method, providing the name of the service as its first parameter, a new service is created and an interface addressing this service is returned by the function.

[21] To get another interface provided by a service in ooRexx, use an existing interface object and send the name of the required interface as a message to this object (like a function). Like calling a method of the interface object, a new interface object representing the required interface is returned. Here a short example: “newinterface = x_Desktop~XDispatchProvider”.

UNO component is responsible for the call. Now call the "`executeDispatch`" method of the "`XDispatchHelper`" interface as shown in the example below.

- Task of the macro: Invoking other macros.
- Peculiarities: How to execute other macros and how to address them.
- Possible solution: usage of "`DispatchProvider`" service.

x_RunMacro.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- this macro just works externally, called by rexxj or rexx

-- create DispatchHelper service and query its interface
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
s_DispatchHelper = x_MultiServiceFactory~createInstance("com.sun.star.frame.DispatchHelper")
x_DispatchHelper = s_DispatchHelper~XDispatchHelper

-- get dispatch provider interface of current Desktop
x_DispatchProvider = x_Desktop~XDispatchProvider
-- define ooRexx dispatch target
MacroURL = "vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=user"

-- prepare parameters
parameters = uno.CreateArray(.UNO~PROPERTYVALUE, 2)
-- traget script ignores argument names, use any name you want
parameters[1] = uno.createProperty("arg1", 5)
parameters[2] = uno.createProperty("arg2", 2)

-- make dispatch call
-- ATTENTION! do not use .nil here, instead use .uno~noProps if no parameters submitted !!!
-- i.e: x_DispatchHelper~executeDispatch(x_DispatchProvider, MacroURL, "", 0, .uno~noProps)
r = x_DispatchHelper~executeDispatch(x_DispatchProvider, MacroURL, "", 0, parameters)

.bsf.dialog~messageBox("Result of x_Sample.rex: " || r~result, "IT Works", "information")

-- define Star Basic dispatch target
MacroURL = "vnd.sun.star.script:BakkMacros.x_Sample.addition?language=Basic&location=application"

-- r = x_DispatchHelper~executeDispatch(x_DispatchProvider, MacroURL, "", 0, .uno~noProps)
r = x_DispatchHelper~executeDispatch(x_DispatchProvider, MacroURL, "", 0, parameters)

.bsf.dialog~messageBox("Result of x_Sample.addition (Star Basic Macro): " || r~result, -
```

```
"IT Works", "information")

::requires UNO.CLS
```

Sourcecode 2: x_RunMacro.rex

References for x_RunMacro.rex:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

PropertyValue array: Cf. [REFOOO03], [REFOOO04], [REFBSF02], [IDLRef12].

Messagebox: Cf. [REFBSF01].

Example specific references: Cf. [UMIFOO03], [IDLRef09], [IDLRef10], [IDLRef11], [REFOOO05].

See sourcecode 3 for the called ooRexx macro.

x_Sample.rex

```
-- a small test macro to test the x_RunMacro.rex macro

info = "Adding: " || ARG(1) || " + " || ARG(2) || " using ooRexx"
.bsf.dialog~messageBox(info, "IT Works", "information")

return (ARG(1) + ARG(2))

::requires BSF.CLS
```

Sourcecode 3: x_Sample.rex

References for x_Sample.rex:

Messagebox: Cf. [REFBSF01].

The example also calls a basic macro, to show how to address a basic macro: Store to "`My Macros`", Library name: "`BakkMacros`", Module: "`x_Sample`":

x_Sample.bas

```
REM  *****  BASIC  *****

Sub Main
   RunMacro
End Sub

Function addition(arg1, arg2 as Integer) as Integer

   ' view that we are currently using Star Basic
   MsgBox("Adding: " & arg1 & " + " & arg2 & " using Star Basic", 64, "IT Works")

   ' return calculation
   ' to calculate make sure the parameters are Integers
   addition = CInt(arg1) + CInt(arg2)

End Function

Sub RunMacro

  ' create the Dispatcher service
  oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
```

```
  ' prepare parameters as array
  Dim a(1) As New com.sun.star.beans.PropertyValue
  a(0).Name = "arg1" : a(0).Value = 7
  a(1).Name = "arg2" : a(1).Value = 1

  ' macro URL to addition function above
sMacroURL = "vnd.sun.star.script:BakkMacros.x_Sample.addition?language=Basic&location=application"

  ' call addition function
  r = oDisp.executeDispatch(StarDesktop, sMacroURL, "", 0, a())

  ' view result
  MsgBox("Result of x_Sample.addition: " & r.result, 64, "IT Works")

  ' macro URL to x_Sample.rex
  sMacroURL = "vnd.sun.star.script:BakkMacros.x_Sample.rex?language=ooRexx&location=user"

  ' call x_Sample.rex and use the same parameters again
  r = oDisp.executeDispatch(StarDesktop, sMacroURL, "", 0, a())

  ' show result
  MsgBox("Result of x_Sample.addition: " & r.result, 64, "IT Works")

End Sub
```

Sourcecode 4: x_Sample.bas

References of x_Sample.bas:

Example specific references: Cf. [UMIFOO03], [IDLRef09], [IDLRef10], [IDLRef11], [IDLRef12].

The visual outputs of these macros:

Figure 11 shows how "`x_Sample.rex`" is called by "`x_RunMacro.rex`" and is displaying a message box.

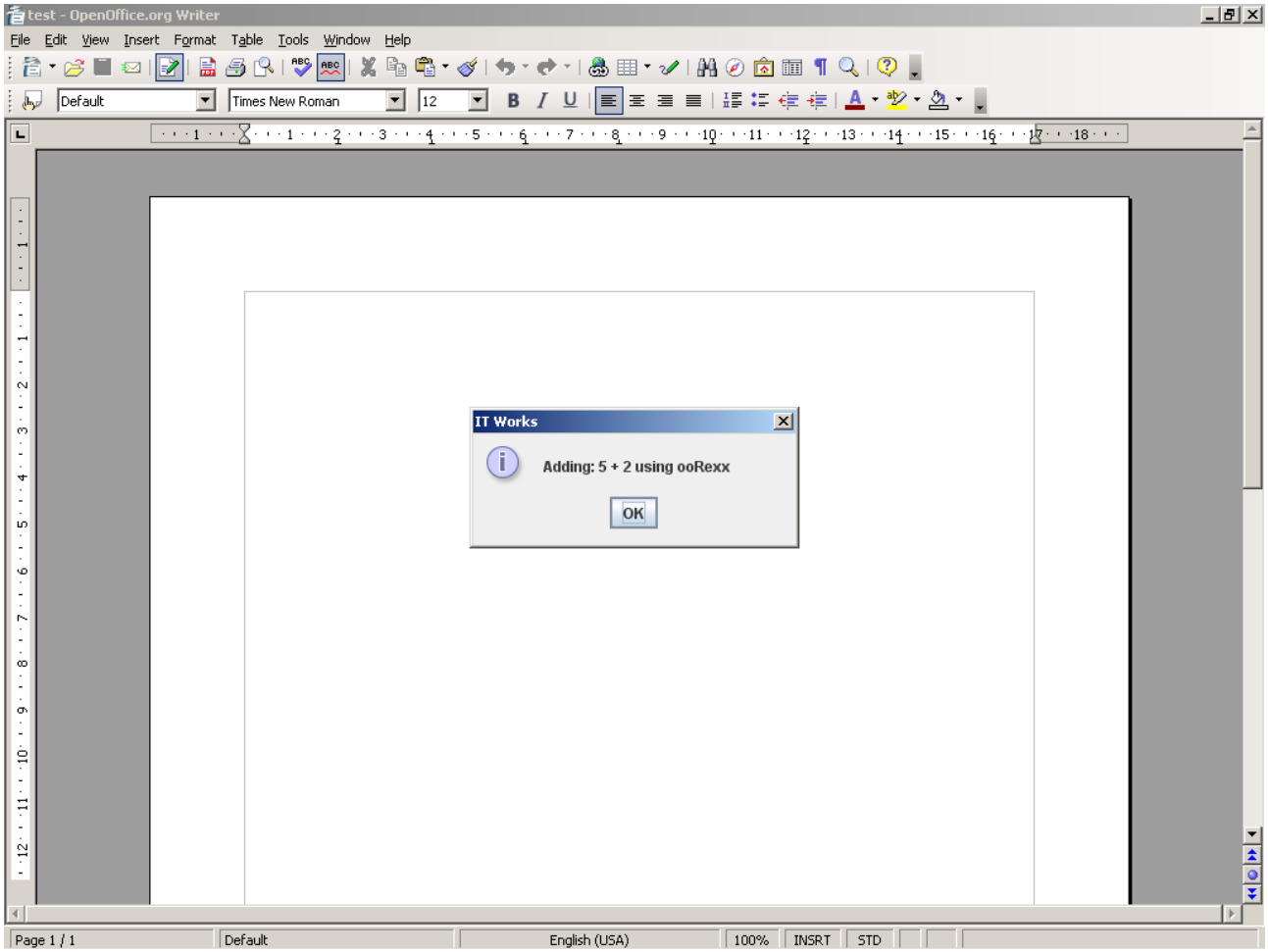

Figure 11: Output of x_Sample.rex.

The next picture, figure 12 shows the messagebox generated by "`x_RunMacro.rex`" after its call to "`x_Sample.rex`".

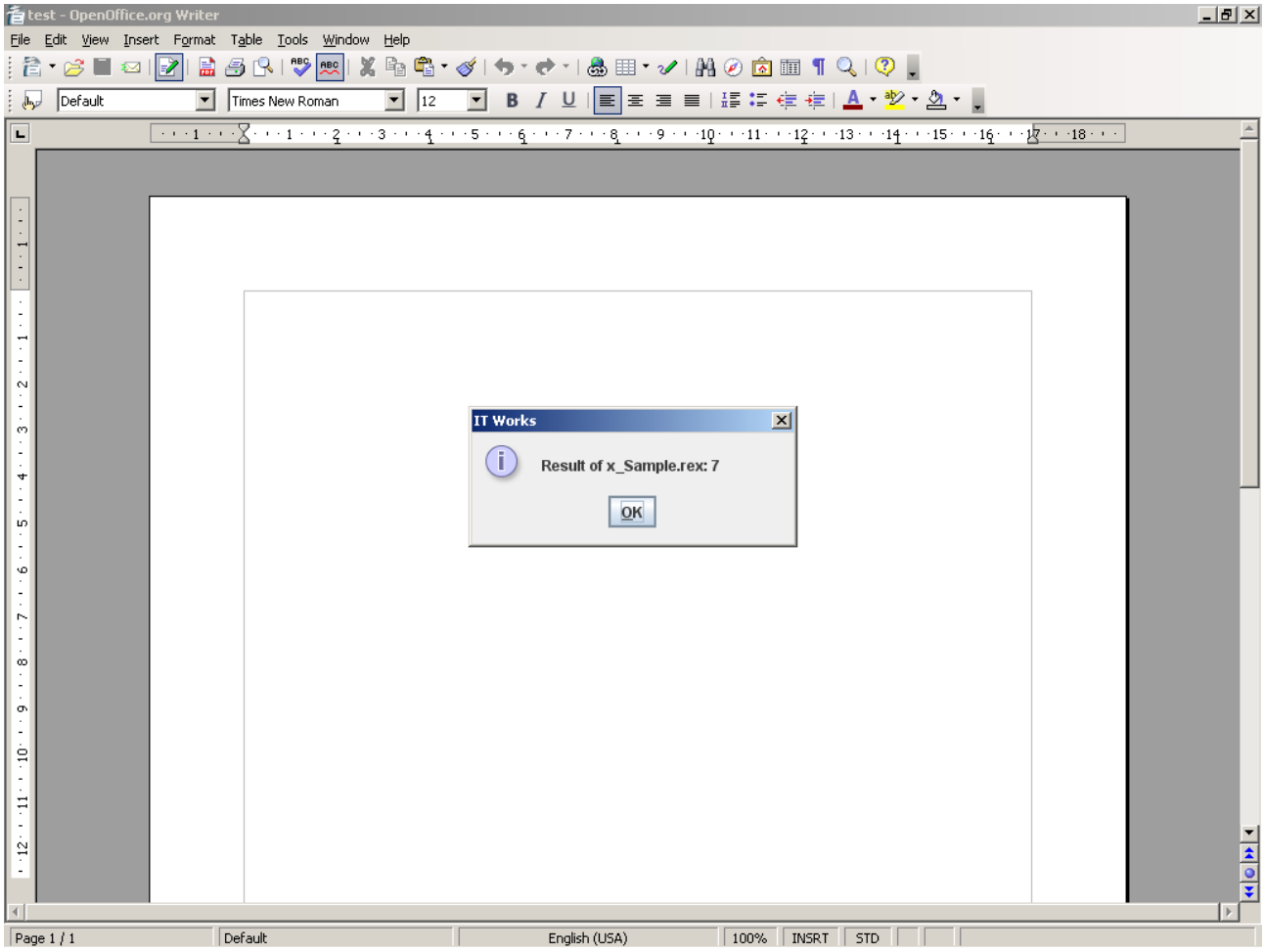

Figure 12: Output of x_RunMacro.rex after calling x_Sample.rex.

Figure 13 shows the messagebox created by the "`addition`" function of the Star Basic example called by "`x_RunMacro.rex`".

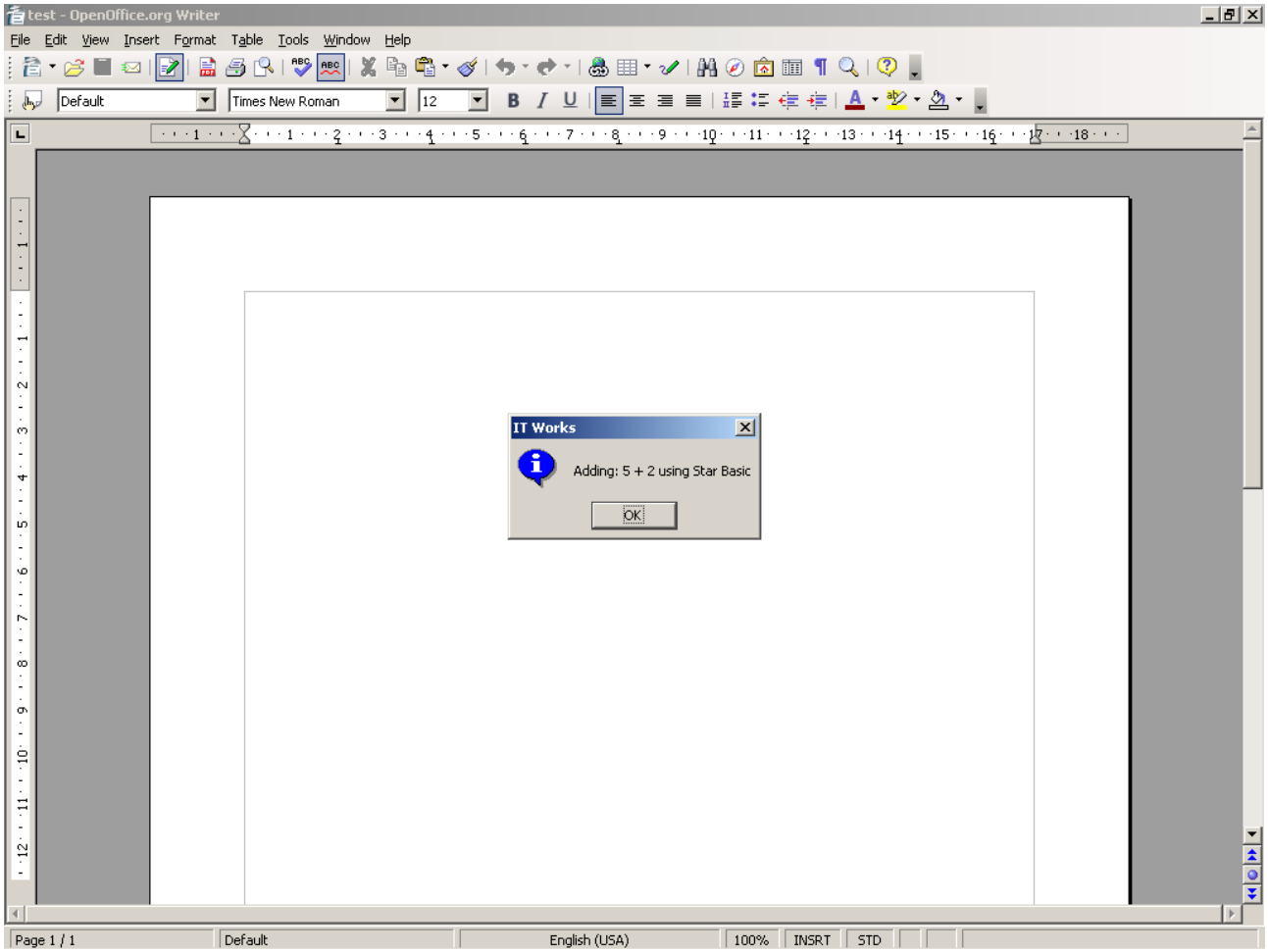

Figure 13: Output of „addition“ function.

Finally, figure 14, a screenshot of the messagebox displayed by the "`x_RunMacro.rex`" macro after calling the Star Basic function.

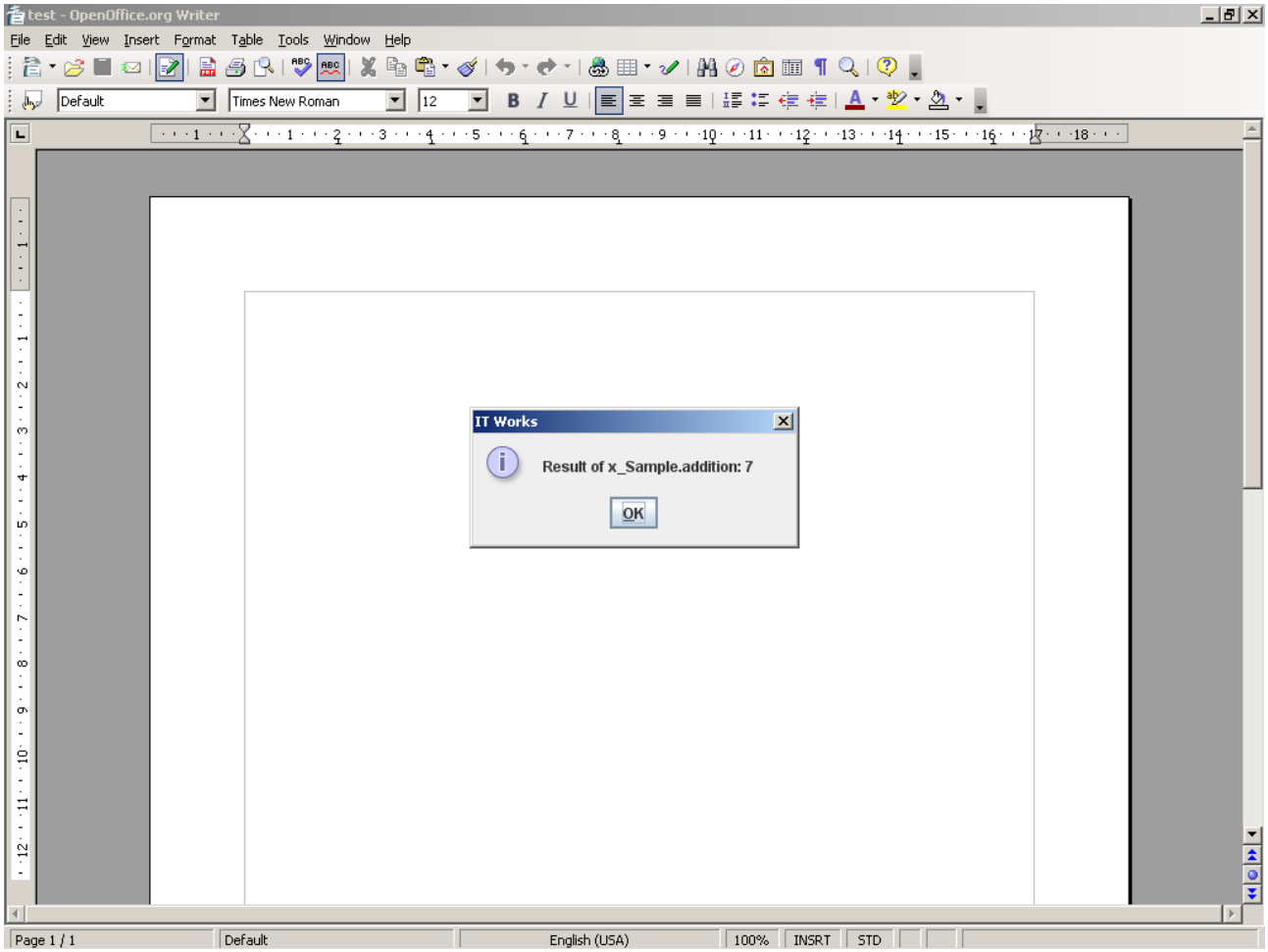

Figure 14: Output of x_RunMacro.rex after call to "addition" function.

The Star Basic example above is also able to call ooRexx macros itself, by calling the "`RunMacro`" method.

The next example shows how to force an ooRexx macro to call another ooRexx macro which resides in the same folder. The problem here is, that if the macro has been started from inside Open Office, the execution path does not correspond to the path where the macro is located. Therefore it is necessary to find out where the macro is located and to set the systems Path environment variable to enable ooRexx to find the other macro. The "`uno.addPath()`" function does these previously described tasks and returns the path of the macro as result. Before the macro ends, the environment variable should be restored to its old value. This is done by calling "`uno.removePath()`" providing the received macro path as a parameter.

- Task of the macro: Invoking other ooRexx macros.
- Peculiarities: How to set the path to let ooRexx find the other ooRexx macros.
- Possible solution: usage of "`uno.addPath()`" and "`uno.removePath()`" methods.

x_RunRexxMacro.rex

```
-- append the current macro path to environmental variable path
addedPath = uno.addPath()

-- call another rexx script
-- call (addedPath"\macroenvironment.rex")
r = x_Sample.rex(1, 4)

.bsf.dialog~messageBox("Result of x_Sample.rex: " || r, "IT Works", "information")

-- remove the current macro path from the environmental variable path
call uno.removePath(addedPath)

::requires UNO.CLS
```

Sourcecode 5: x_RunRexxMacro.rex

References of this macro:

Messagebox: Cf. [REFBSF01].

Example specific references: Cf. [REFOOO07], [REFOOO08].

Visual output of this macro:

Figure 15 shows the message box displayed after "`x_Sample.rex`" was called by "`x_RunRexxMacro.rex`".

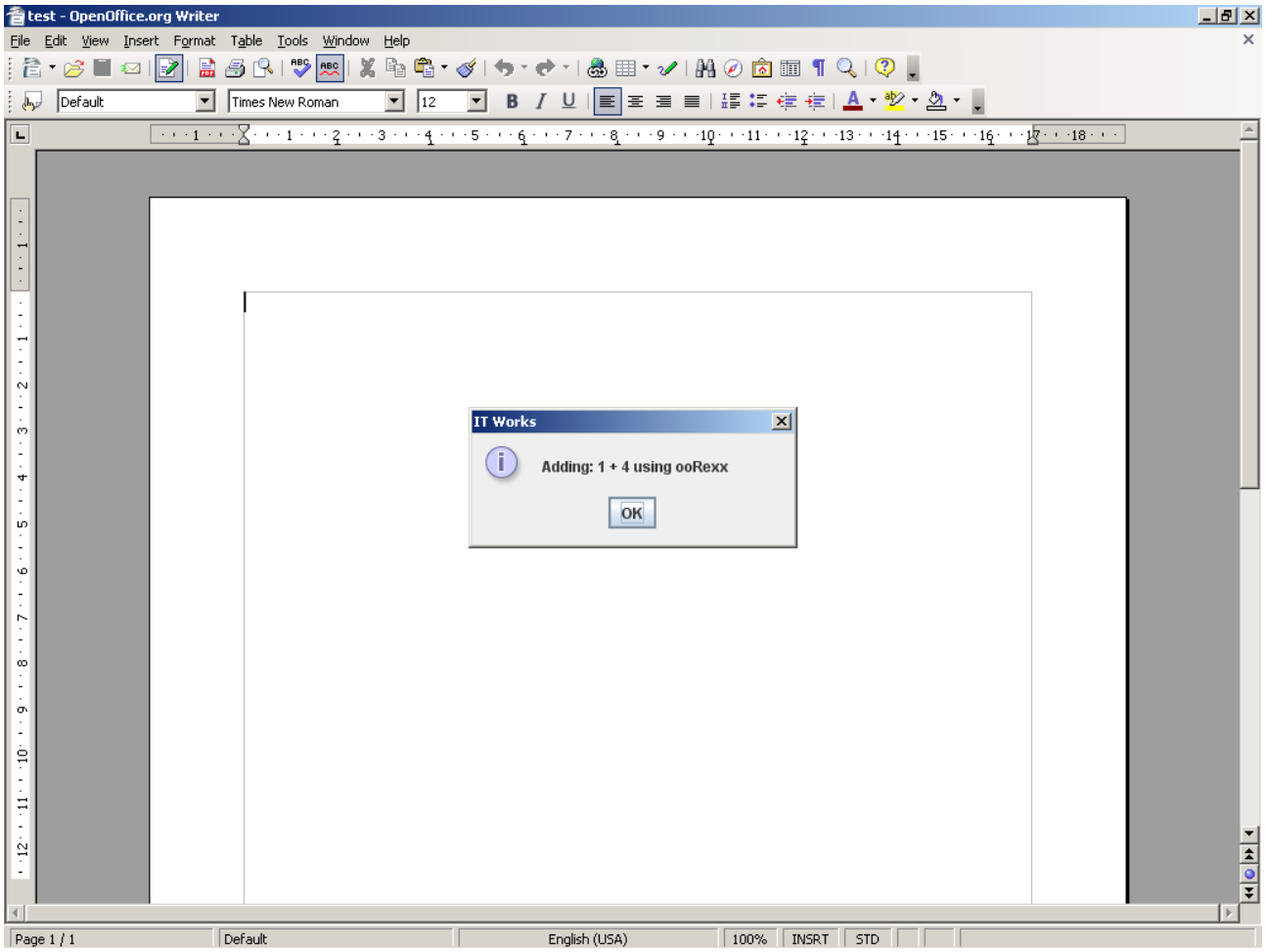

Figure 15: Output of x_Sample.rex called by x_RunRexxMacro.rex.

The output of "`x_RunRexxMacro.rex`" after calling "`x_Sample.rex`" is depicted in figure16 below.

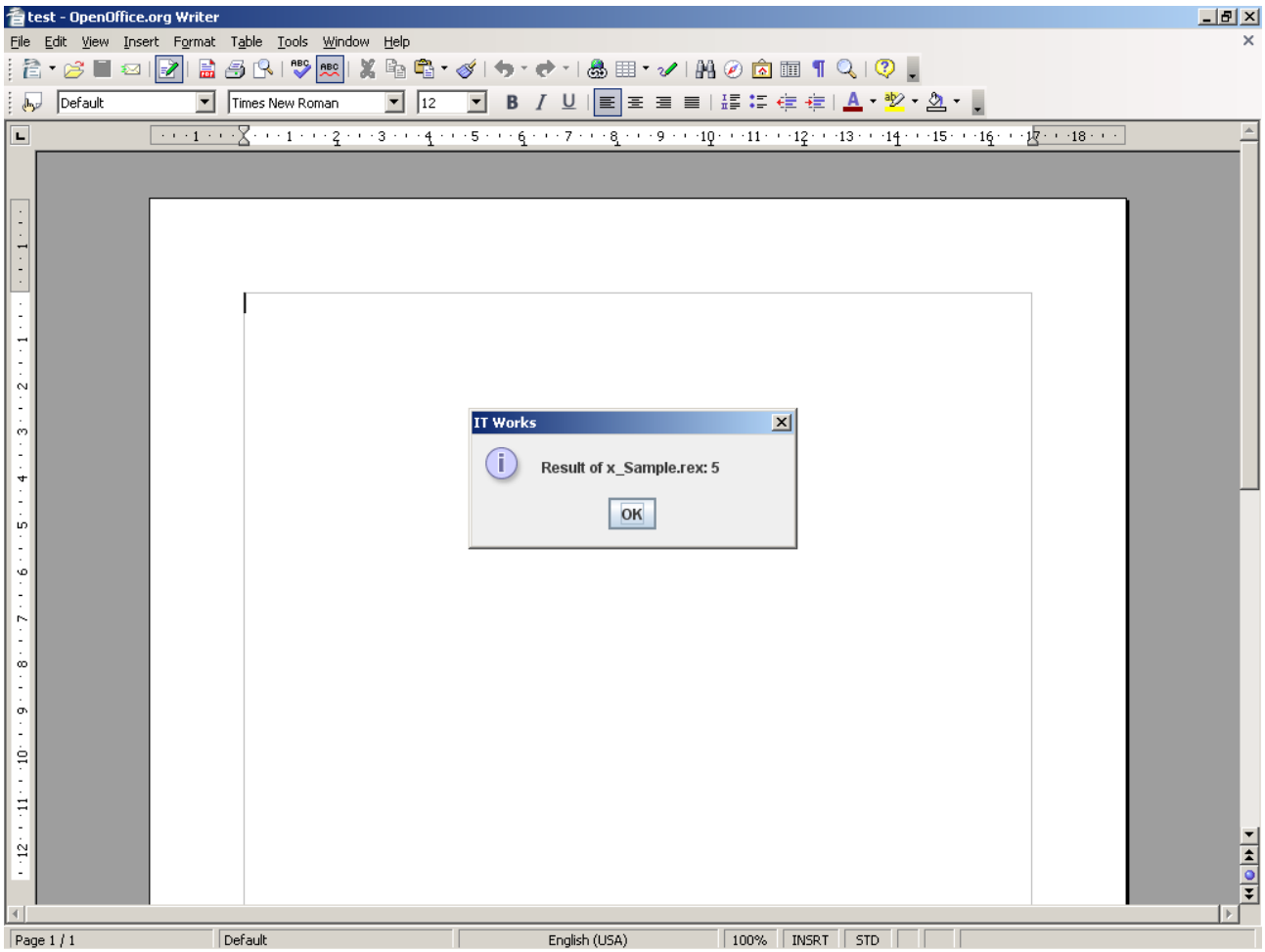

Figure 16: Output of x_RunRexxMacro.rex.

Conclusion

The first method in this chapter is able to call all the macros of all languages, as long as they are registered in a description[22] file and located in one of the storage paths of macros known by Open Office. The second method only enables us to call other ooRexx macros, which reside in the same folder as the calling macro, but the called macro does not need to be stated within the description file of the library[23].

[22] The description file is named "`PARCEL-DESCRIPTOR.XML`".
[23] It is "hidden" from the user, if he is just using Open Office Dialogs to start or edit a macro.

5 How to Connect to Open Office

When starting a macro from outside of Open Office, a connection must be established using the UNO environment. To connect to Open Office also the URL, which has been used to start the listening server, must be known in order to identify Open Office as the target of the connection. Furthermore the initial factory service, capable of creating all the other services, needs to be identified. This Service is named "`ServiceManager`", and must be appended to the URL. Finally the connection URL looks something like this:

"`uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager`".

To establish a connection some context, called component context, is needed to hold all the UNO objects and services. The creation of this component context requires the bootstrap class provided by the Java bridge, which is able to instantiate such a context object. By calling the "`getServiceManager`" method of the component context object an UNO "`ServiceManager`" is created which allowes registering an URL connection object. With the connection object a connection can be setablished by calling its resolve method assigning the connection URL as a parameter. The return value of this method is containing the service manager object of Open Office, which gives access to many services of Open Office.

This is a small example of the described connection method above:

- Task of the macro: Establish a connection to Open Office.
- Peculiarities: creating a Component Context and establish a network connection.
- Possible solution: usage of "`Bootstrap`" class and "`UnoUrlResolver`" service. Easier solution: usage of "`uno.connect()`".

x_LargeConnect.rex

```
-- identify target program
unourl = "uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager"

-- create bootstrap object
bootstrap = uno.wrap(.bsf~new("com.sun.star.comp.helper.Bootstrap"))

-- create the Component Context
x_ComponentContext = bootstrap~createInitialComponentContext(.nil)

-- get the UNO Service Manager
s_UnoServiceManager = x_ComponentContext~getServiceManager()

-- create an UNO connection interface
urlresolver = "com.sun.star.bridge.UnoUrlResolver"
s_UnoUrlResolver = s_UnoServiceManager~createInstanceWithContext(urlresolver, x_ComponentContext)
x_UnoUrlResolver = s_UnoUrlResolver~XUnoUrlResolver
```

```
-- connect to Open Office and retrieve "StarOffice.ServiceManager" service
s_ServiceManager = x_UnoUrlResolver~resolve(unourl)

::requires UNO.CLS
```

Sourcecode 6: x_LargeConnect.rex

References of this macro:

Example specific references: Cf. [ProfUNO03], [JAVAREF01], [IDLRef02], [IDLRef13], [IDLRef14].

BSF4Rexx also provides a module called "`UNO.CLS`" which contains many methods to ease programing ooRexx macros for Open Office. One method is able to do the whole connection by a single call. This method will return a component context object already connected to Open Office and also containing the Open Office service manager object. Using this method it is not necessary to supply a connection URL, because the method will use its own URL[24].

An example:

- Task of the snippet: Establish a connection to Open Office.
- Peculiarities: creating a Component Context, the Desktop service and get the last opened document.
- Possible solution: usage of "`uno.connect()`".

```
-- called from outside of OOo, create a connection

-- connect to Open Office and get component context
x_ComponentContext = UNO.connect()
-- create a desktop service and its interface
service = "com.sun.star.frame.Desktop"
s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
x_Desktop = s_Desktop~XDesktop
-- get the last active document
x_Document = x_Desktop~getCurrentComponent()
```

Sourcecode 7: Connect and Prepare macro outside of Open Office

References:

Connection: Cf. [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO02].

When running the macro from inside Open Office, no connection is required. Instead a "`ScriptContext`" object is available. "`UNO.CLS`" also provides a method to get this object. The "`ScriptContext`" object allows getting the component context object, as well as the macros desktop and document interface.

[24] It tries to connect to port 2002.

An example for the usage of "`ScriptContext`":

- Task of the snippet: Get the access to the macro environment, when the macro was called inside Open Office.
- Peculiarities: creating a Component Context, the Desktop service and get the opened document.
- Possible solution: usage of "`uno.getScriptContext()`".

```
x_ScriptContext = uno.getScriptContext()
-- invoked by OOo as a macro

-- get context
x_ComponentContext = x_ScriptContext~getComponentContext
-- get desktop (an XDesktop)
x_Desktop = x_ScriptContext~getDesktop
-- get current document
x_Document = x_ScriptContext~getDocument
```

Sourcecode 8: Prepare Macro inside Open Office

References:

Script Context: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [REFOOO01].

A Combination of the last two methods would allow the macros to be called either inside or outside of Open Office. To do so first try to get a "`ScriptContext`". If no "`ScriptContext`" is present a connection must be established, otherwise we just use the script context to get the component context.

Here the combination Example:

This code can be found at the beginning of nearly every example provided in this paper:

- Task of the snippet: Establish a connection to Open Office.
- Peculiarities: creating a Component Context, the Desktop service and get the last opened[25] or the related opened document[26].
- Possible solution: usage of "`uno.getScriptContext()`" and "`uno.connect()`".

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if(x_ScriptContext<>.nil)then
do
 -- invoked by OOo as a macro

 -- get context
 x_ComponentContext = x_ScriptContext~getComponentContext
```

[25] If the macro was called from outside Open Office.
[26] If the macro was run inside Open Office.

```
 -- get desktop (an XDesktop)
 x_Desktop = x_ScriptContext~getDesktop
 -- get current document
 x_Document = x_ScriptContext~getDocument
end
else
do
 -- called from outside of OOo, create a connection

 -- connect to Open Office and get component context
 x_ComponentContext = UNO.connect()
 -- create a desktop service and its interface
 service = "com.sun.star.frame.Desktop"
 s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
 x_Desktop = s_Desktop~XDesktop
 -- get the last active document
 x_Document = x_Desktop~getCurrentComponent()
end
```

Sourcecode 9: Header of this papers macros

References:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

6 Examples

Here are now several examples on how to deal with Open Office by using ooRexx macros. The examples are divided into three categories: environment related examples, writer examples, and calc examples.

6.1 Examples Related to the Environment of Open Office

The following examples describe some functionality of Open Office, which is independend of any Program of the Open Office package. The example 6.1.5 is using the Writer Program for output only.

6.1.1 Example 1: Closing a Document

- Task of the macro: Close an opened document.
- Peculiarities: find the best way to close the document.
- Possible solution: usage of "`XClosable`", "`XComponent`" and "`XDesktop`" interfaces.

This example shows different ways to close a document. There are three ways to close a document: The "`close`" method of the "`XClosable`" interface, the "`dispose`" function of the "`XComponent`" interface and the "`terminate`" method of the "`XDesktop`" interface. The script will try to use the most softest way possible. The best way to close a document is to use the "`close`" method of the "`XClosable`" interface in case the document object inherits this interface. Calling the close method allows the document to disagree and cancel the termination, if it is not ready for it. The second way to close a document is to call the "`dispose`" function of its "`XComponent`" interface. Using this method will assure that this component is terminated regardless of the component being ready to be terminated or not. The third way is quite rough. It is the termination command of the desktop service. It will close all components registered to this desktop, therefore it might also affect other opened documents too. Using this method will also close the desktop service itself.

Additionally this example checks if the document which shall be closed has already been stored to a file before and was modified since then. If that is the case, the macro will save the document before closing it.

e_CloseDocument.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- create outputtext
output = "Document "

-- check if doucument has been changed
x_Modifiable = x_Document~XModifiable
x_Storable = x_Document~XStorable

If (x_Modifiable~isModified()) Then
do
   output = output || "was Modified, "

   /*
   if there is allready a file containing the document
   then save into this file, else just set modify flag to false
   (do not save file)
   */

   If (x_Storable~hasLocation & (\ x_Storable~isReadOnly)) Then
   do
      x_Storable~store()
      output = output || "and has been stored - "
   end
   else
   do
      x_Modifiable~setModified(.false)
      output = output || "and has NOT been stored - "
   end
end
else
do
   output = output || "was NOT Modified - "
end

/*
next check for different methods to shut down
if we are able to create a XModel interface then also try to query a XClosable
interface to close document. If XCloseable interface is not available, use
Documents dispose method to close the document. If XModel interface query fails,
terminate the frame to shut down.
*/

-- x_ServiceInfo = x_Document~XServiceInfo
-- If x_ServiceInfo~supportsService("com.sun.star.frame.XModel") then
-- I dont know why, but this does not work properly, therefore

x_Model = x_Document~XModel

if x_Model <> .nil then
```

```
do
   x_Closeable = x_Document~XCloseable

   If x_Closeable <> .nil then
   do
      x_Closeable~close(.true)
      output = output || "closed by XClosable Interface (SOFTEST WAY)"
   end
   else
   do
      x_Document~dispose()
      output = output || "closed by XDocument Interface (SOFT WAY)"
   end

end
else
do
   x_Desktop~terminate()
   output = output || "closed by XDesktop Interface (HARD WAY)"
end

-- finaly show message what happened
.bsf.dialog~messageBox(output, "Closing Document...", "information")

::requires UNO.CLS
```

Sourcecode 10: e_CloseDocument.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO04].

XModel interface: Cf. [IDLRef17].

Example specific references: Cf. [IDLRef15], [IDLRef16], [IDLRef18].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

Figure 17 show one probable visual output of this macro.

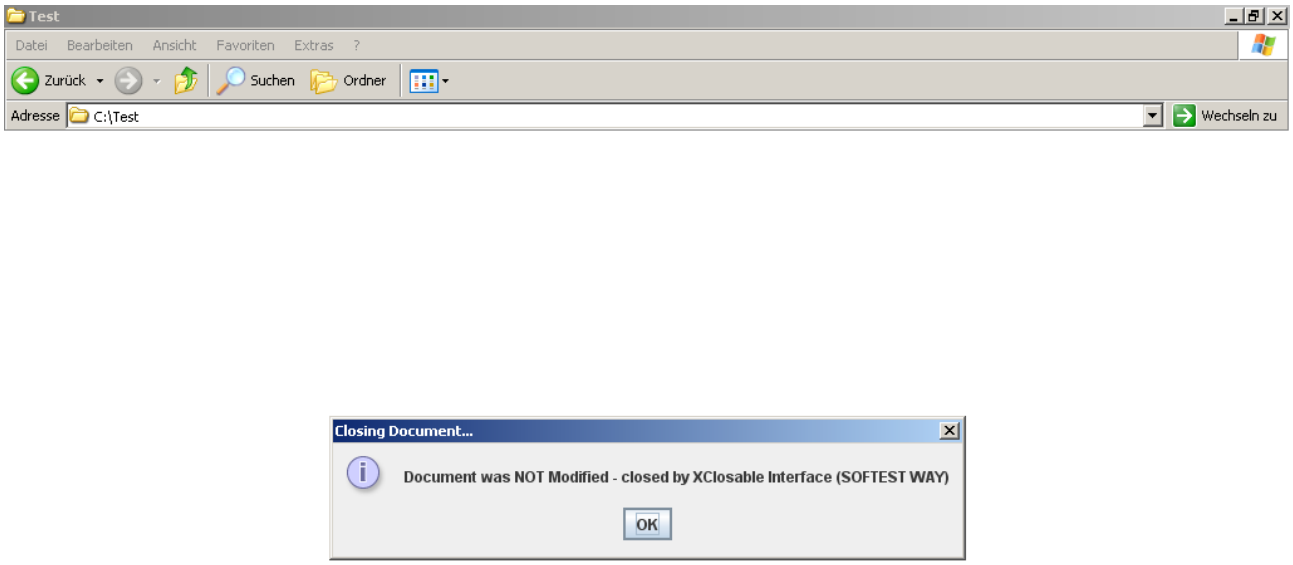

Figure 17: Output of e_CloseDocument.rex.

6.1.2 Example 2: Change the Work Path

- Task of the macro: change the users work path.
- Peculiarities: get access to the path service.
- Possible solution: usage of "`PathService`" service.

This example changes the working path[27] of Open Office. The changes can be observed as path changes of the documents load or save dialog. To achieve this, access to the path settings of Open Office is provided by creating a "`PathService`" service. This service contains all the path settings accessible as a Propertyset. The Open Office API provides a

[27] This is the path to the folder where the user is working at. It is used, for example, if a save dialog is shown and a newly created document shall be stored.

"`XPropertyset`" interface to access such Propertysets. The next step is to write the new path to the property named "`Work`". This will change the current working path.

Additionally this example is using a folder picker dialog to let the user choose the new path. This dialog is a service which has to be instantiated. Creating this service, grants the ability to change the behavior and appearance of the dialog. By calling the "`setDescription`" method the description label of the dialog can be changed. To change the initially viewed path of the dialog to the Open Office work path before it is overwritten, it is necessary to call the "`setDisplayDirectory(workdir)`" method, where "`workdir`" contains the value of the "`Work`" property.

Figure 18 shows the path selection of the macro.

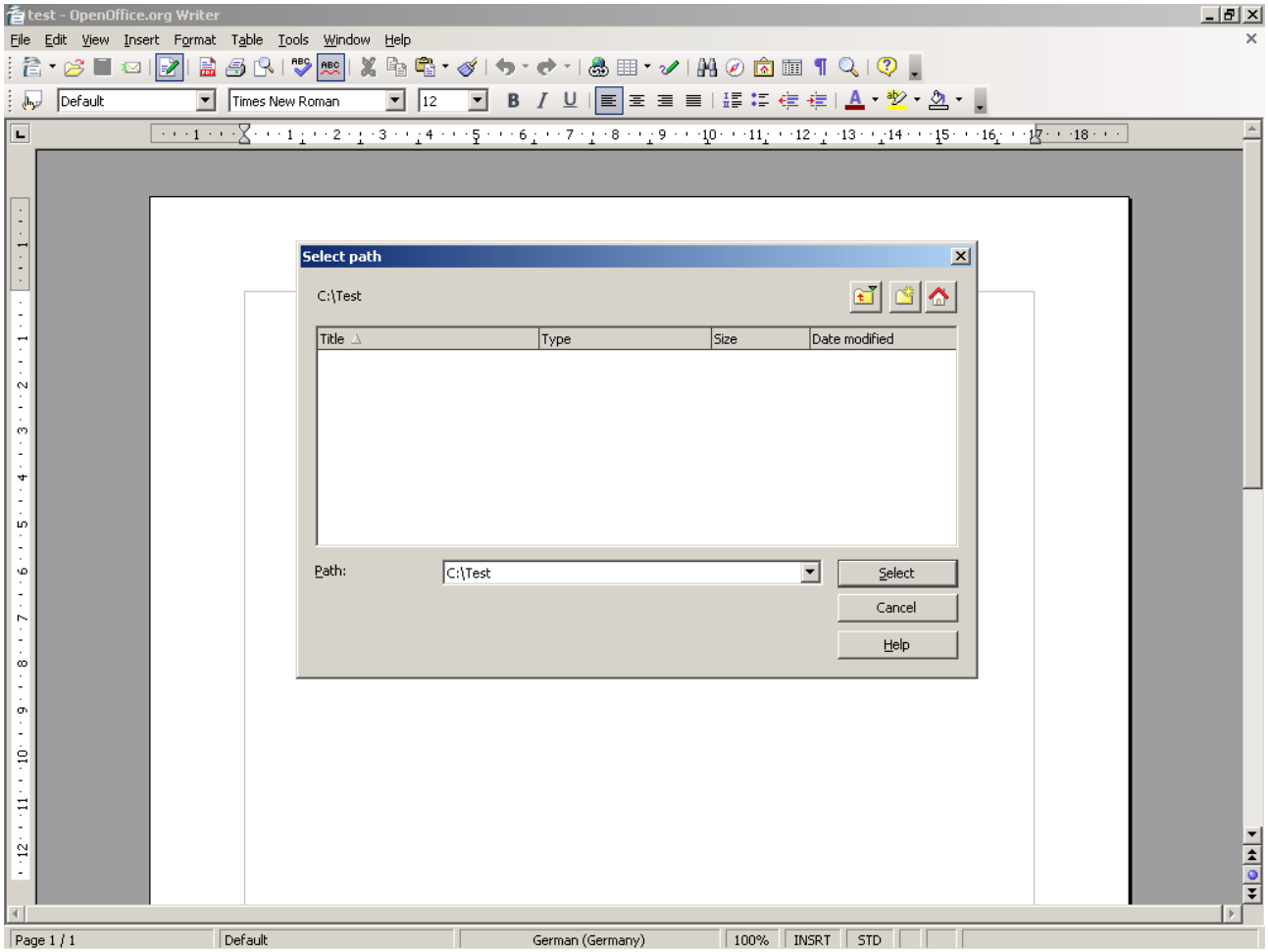

Figure 18: Path selection of e_Path.rex.

However, Open Office 2.4 seems to have problems changing the viewed directory of Windows XP native folder and file picker dialogs. Until this problem has been solved the us-

age of Open Office own dialogs is recommended. One way tho achieve this is to use the default service[28], as documented in the Open Office API documentation and to switch to Open Office own dialogs in the options menu. The examples, using such dialogs, will always use Open Offices own dialogs by directly instantiating the right service. This will override the option setting. The previously mentioned dialog problem also occurs with usage of the "`FilePicker`" dialog, and is solved the same way.

e_Path.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- create the pathservice
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
s_path = x_MultiServiceFactory~createInstance("com.sun.star.util.PathSettings")

-- access it by propertysets
pathproperties = s_path~XPropertySet

-- read the work entry
workdir = pathproperties~getPropertyValue("Work")

-- create a Folder Picker dialog
s_FolderDialog =
x_MultiServiceFactory~createInstance("com.sun.star.ui.dialogs.OfficeFolderPicker")
x_FolderDialog = s_FolderDialog~XFolderPicker

-- Better name for our dialog:
x_FolderDialog~setDescription("Current Workdir: " || workdir)
x_FolderDialog~setDisplayDirectory(workdir)

pathchoosen = x_FolderDialog~execute()

-- if a path has been choosen write the new path into the
-- pathsettings service of Open Office
if ( pathchoosen ) then
do
   librarypath = x_FolderDialog~getDirectory()
   pathproperties~setPropertyValue("Work", librarypath)
end

::requires UNO.CLS
```

[28] The default services name is "`COM.SUN.STAR.UI.DIALOGS.FOLDERPICKER`".

Sourcecode 11: e_Path.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO05].

Example specific references: Cf. [IDLRef19].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Folder Picker: Cf. [IDLRef22], [IDLRef23].

After executing this macro:

Figure 19 shows how the changed Workpath affects the save dialog.

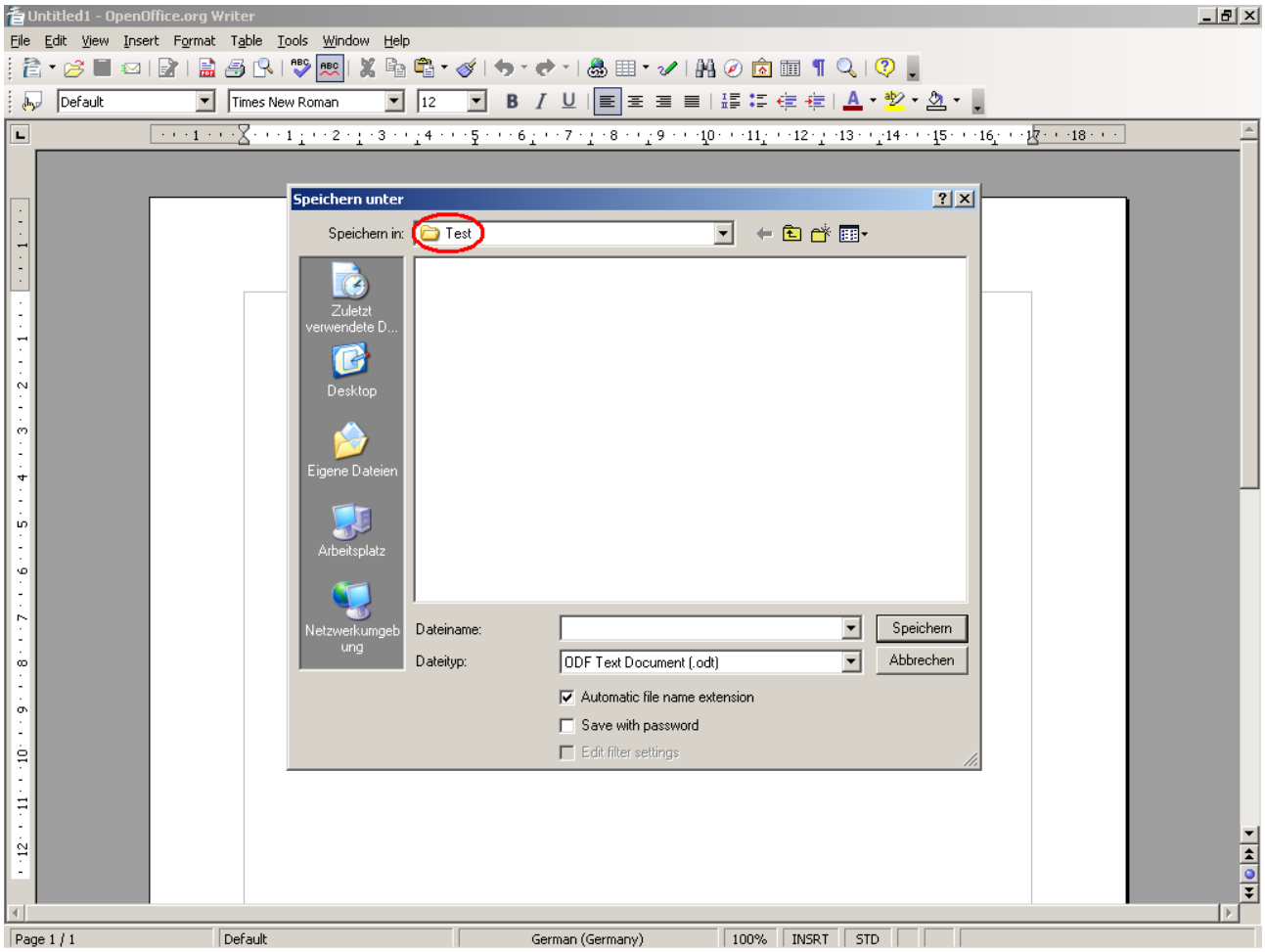

Figure 19: Work path changed.

6.1.3 Example 3: Create a Tool Bar

- Task of the macro: create a tool bar for writer documents with two buttons, which will call the "`w_CreateStyleCode.rex`" and the "`w_ImportCode.rex`" macros.
- Peculiarities: get access to the user interface service and addressing another macro.
- Possible solution: usage of "`ModuleUIConfigurationManagerSupplier`" service.

The first step to create a tool bar is to build a tool bar URL, which is defining the location of storage. It is also an unique identification of the tool bar, so when removing the tool bar, as shown by the next example below, the tool bar can be specified . Next a service capable of manipulating the user interface of Open Office is needed. It is called "`ModuleUIConfigurationManagerSupplier`". Because not every tool bar can be used with every document type[29], access to the document specific user interface manager, which manages the tool bars, must be obtained. By calling "`createSettings`" method of this service a new tool bar is created. Every tool bar is an enumerated container of buttons. To define a button a PropertyValue array must be created. The "`PropertyValue`" is an Open Office API object able to hold values which are related to a specific name. Now this "`PropertyValue`" array is used to configure our buttons. The "`CommandURL`" entry is containing the macro to be executed. About creating macro URLs see Chapter 4.4. The PropertyValue named "`Label`" states the text of the button. "`Type`" defines the type of the entry and the "`Visible`" entry, containing a true value, makes the button appearing. After the button has been configurated it must be added to the tool bar container by using "`insertByIndex`" command. The first parameter of this method is defining the position and the second one the "`PropertyValue`" array of our button. If all buttons have been added successfully, the final task is to add the newly created tool bar to the document specific user interface manager. The problem here is to find out whether this tool bar has already been registered or not. This information can be retrieved by the return value of the "`hasSettings`" function. If the tool bar is registered then use "`replaceSettings`" method otherwise use "`insertSettings`" to register the new tool bar. Both methods require the tool bar URL as first parameter and the tool bar container as second parameter.

[29] i.e.: Calculation settings for Writer.

The tool bar in this example provides two buttons calling the macros described in chapter 6.2.1 and 6.2.2.

e_CreateToolbar.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- define where to store the toolbar
ToolbarURL = "private:resource/toolbar/custom_exampletoolbar"

-- get the user interface configuration
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
configsupplier = "com.sun.star.ui.ModuleUIConfigurationManagerSupplier"
s_Supplier = x_MultiServiceFactory~createInstance(configsupplier)
x_Supplier = s_Supplier~XModuleUIConfigurationManagerSupplier

-- the document type this toolbar is bound to
DocumentType = "com.sun.star.text.TextDocument"

-- get the user interface configuration of writer
x_UIConfigurationManager = x_Supplier~getUIConfigurationManager(DocumentType)

-- create a new toolbar for writer
x_IndexContainer = x_UIConfigurationManager~createSettings()

-- configure toolbar
-- set name of toolbar
x_Propertyset = x_IndexContainer~XPropertySet
x_Propertyset~setPropertyValue("UIName", "Bakk Statusbar")

-- configure toolbar element
DefaultButton = bsf.getConstant("com.sun.star.ui.ItemType", "DEFAULT")

toolbarbutton = uno.CreateArray(.UNO~PROPERTYVALUE, 4)

MacroURL = "vnd.sun.star.script:BakkMacros.w_CreateStyleCode.rex?language=ooRexx&location=user"
toolbarbutton[1] = uno.createProperty("CommandURL", MacroURL)
toolbarbutton[2] = uno.createProperty("Label", "Create Style: code")
toolbarbutton[3] = uno.createProperty("Type", DefaultButton)
toolbarbutton[4] = uno.createProperty("Visible", .true)

-- add toolbar element
x_IndexContainer~insertByIndex(0, toolbarbutton)

-- configure another toolbar element
MacroURL = "vnd.sun.star.script:BakkMacros.w_ImportCode.rex?language=ooRexx&location=user"
toolbarbutton[1] = uno.createProperty("CommandURL", MacroURL)
toolbarbutton[2] = uno.createProperty("Label", "Import Code from GVim")
toolbarbutton[3] = uno.createProperty("Type", DefaultButton)
```

```
toolbarbutton[4] = uno.createProperty("Visible", .true)

-- and add it at second position
x_IndexContainer~insertByIndex(1, toolbarbutton)

-- if the toolbar allready exists replace it, otherwise add it to the user interface
If x_UIConfigurationManager~hasSettings(ToolbarURL) then
do
   x_UIConfigurationManager~replaceSettings( ToolbarURL, x_IndexContainer )
end
else
do
   x_UIConfigurationManager~insertSettings( ToolbarURL, x_IndexContainer )
end

::requires UNO.CLS
```

Sourcecode 12: e_CreateToolbar.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO06].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Example specific references: Cf. [IDLRef24], [IDLRef25], [IDLRef26], [IDLRef27] [IDLRef28].

getConstant(): Cf. [REFBSF03].

PropertyValue array: Cf. [REFOOO03], [REFOOO04], [REFBSF02], [IDLRef12].

Screenshots of the result of this macro:

If the macro has been executed sucessfully, the newly created tool bar appears as depicted in figure 20.

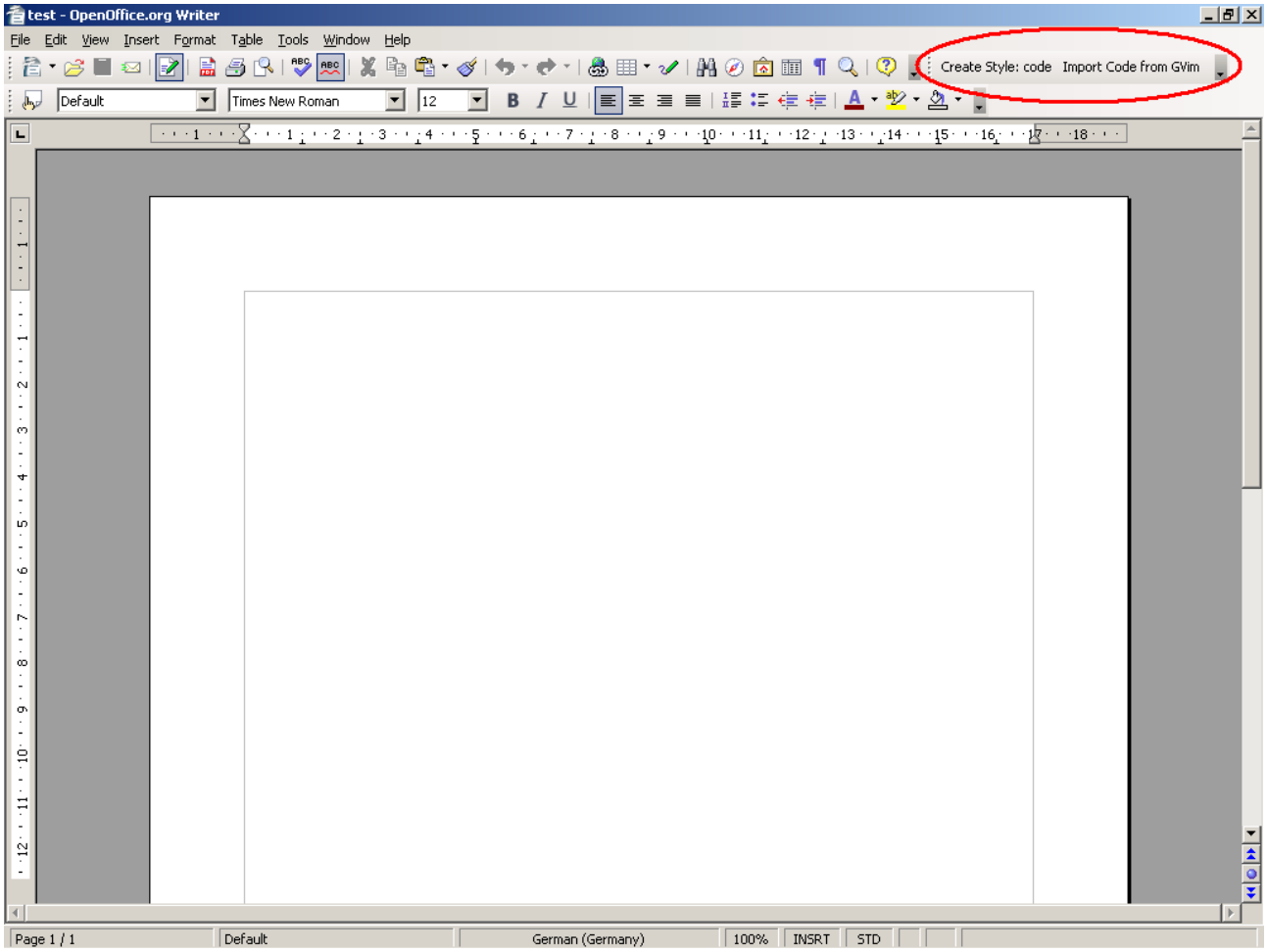

Figure 20: Tool bar created.

Pressing a button very often, might result in crashing Open Office.

6.1.4 Example 4: Remove the Tool Bar

- Task of the macro: remove the tool bar which was created by the previous example.
- Peculiarities: get access to the user interface service and identifying the right tool bar.
- Possible solution: usage of "`ModuleUIConfigurationManagerSupplier`" service.

To remove the previously added tool bar its URL, the "`ModuleUIConfigurationManagerSupplier`" service and its document specific user interface manager have to be retrieved. Before removing the tool bar from the user interface make sure it has been registered before by using the "`hasSettings`" function. If the tool bar is registered the "`removeSettings`" method is able to be called without getting an error. This method requires the tool bar URL as its only parameter to remove the tool bar.

e_RemoveToolbar.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- define toolbar storage position
ToolbarURL = "private:resource/toolbar/custom_exampletoolbar"

-- create the user interface supplier
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
configsupplier = "com.sun.star.ui.ModuleUIConfigurationManagerSupplier"
s_Supplier = x_MultiServiceFactory~createInstance(configsupplier)
x_Supplier = s_Supplier~XModuleUIConfigurationManagerSupplier

-- Specify the document type (writer)
DocumentType = "com.sun.star.text.TextDocument"

-- next get the manager
x_UIConfigurationManager = x_Supplier~getUIConfigurationManager(DocumentType)

-- check if such a toolbar exists
If x_UIConfigurationManager~hasSettings(ToolbarURL) then
do
   -- if it exists remove it
   x_UIConfigurationManager~removeSettings(ToolbarURL)
   .bsf.dialog~messageBox("Toolbar removed", "RemoveToolbar.rex", "information")
end
else
do
   -- outherwise just send error message
   .bsf.dialog~messageBox("Toolbar not installed", "RemoveToolbar.rex", "error")
```

```
end

::requires UNO.CLS
```

Sourcecode 13: e_RemoveToolbar.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO06].

Example specific references: Cf. [IDLRef24], [IDLRef25], [IDLRef26].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

Figure 21 shows the GUI output, after calling the macro,

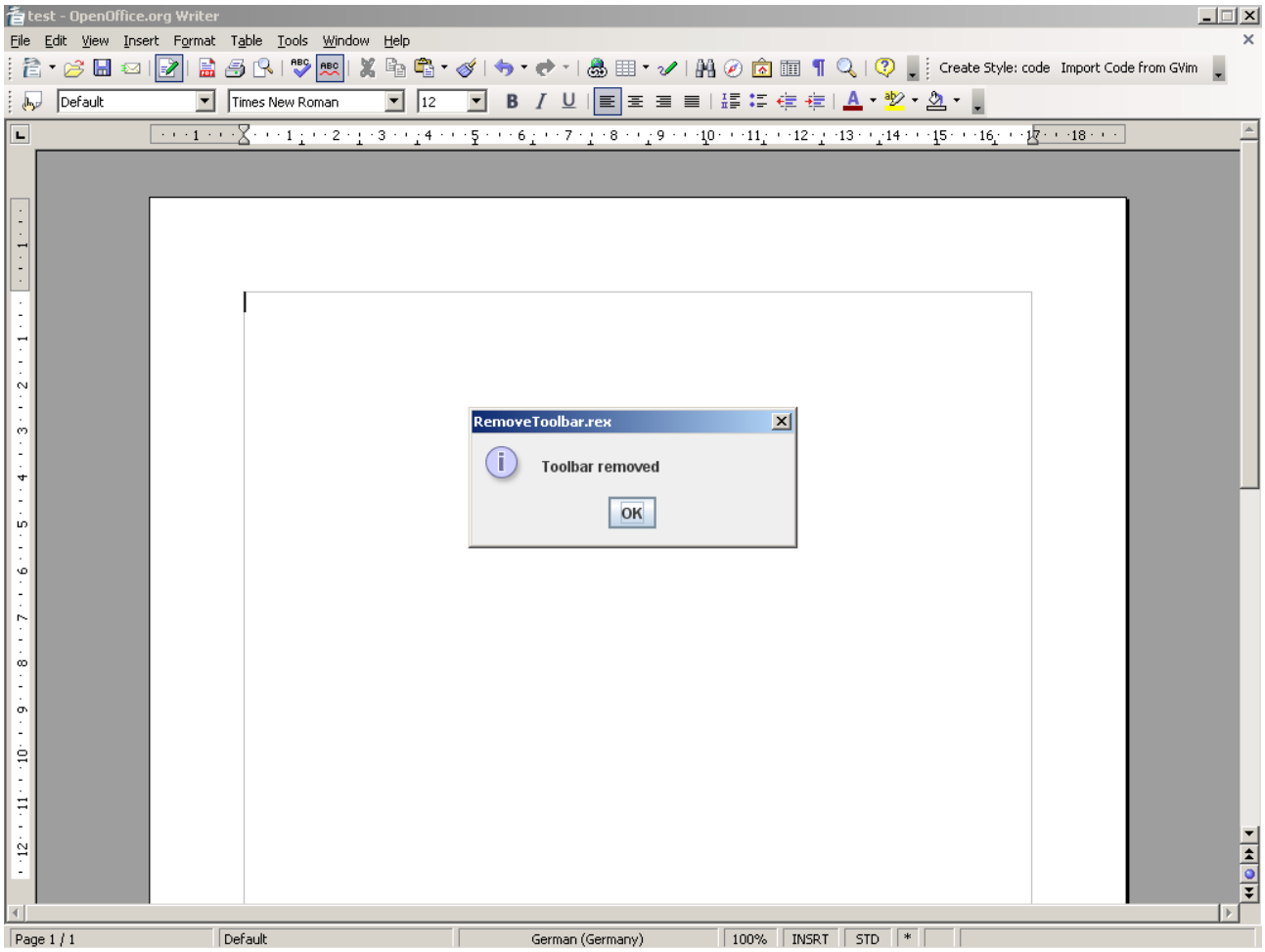

Figure 21: Message box of e_RemoveToolbar.rex.

and figure 22 shows, that the tool bar has been removed:

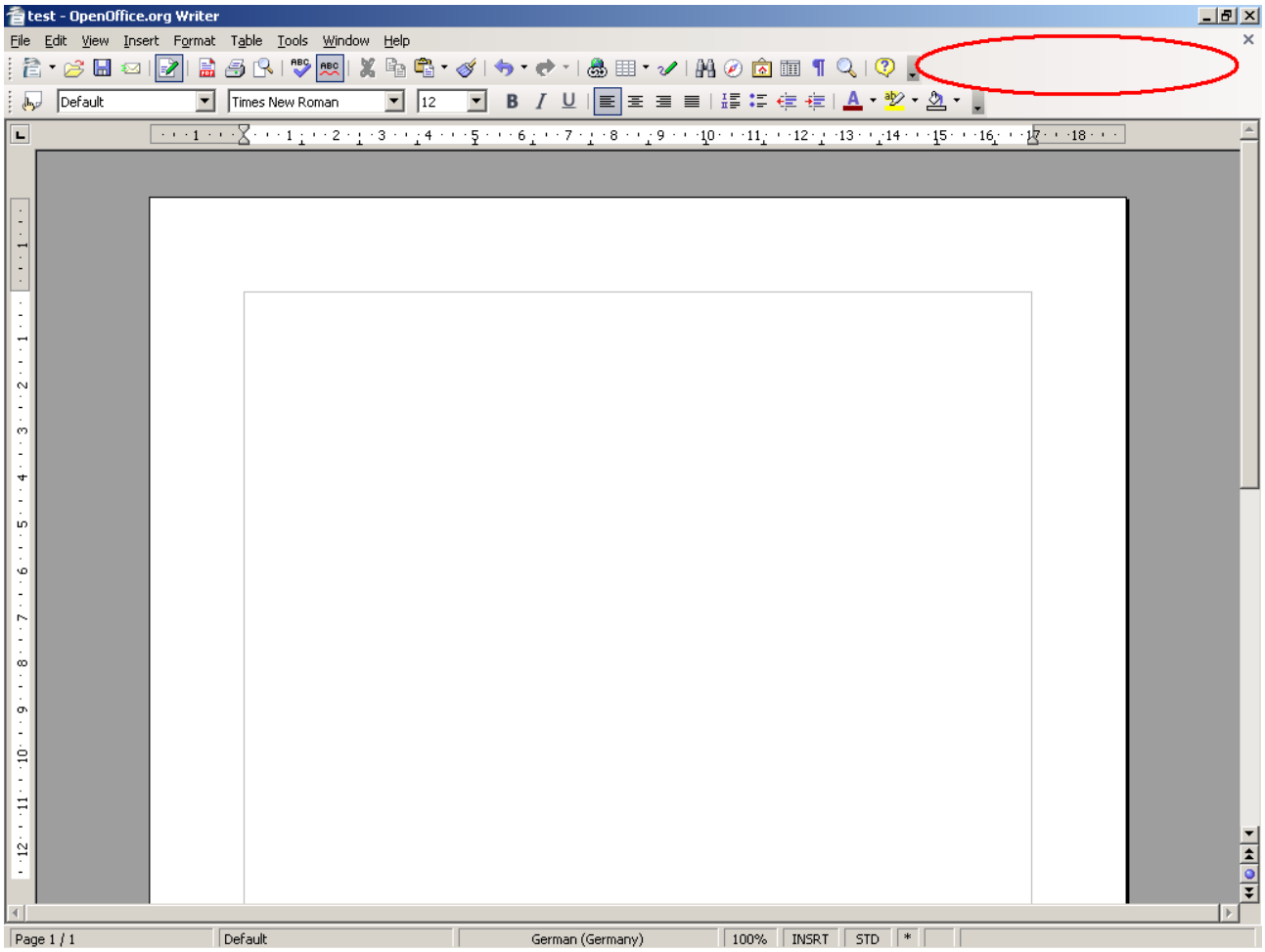

Figure 22: Tool bar disappeared.

After the toolbar has been removed, Open Office has problems to recreate a toolbar. Trying this will produce an empty toolbar, which cannot be removed, but closing Open Office.

6.1.5 Example 5: Read the System Clipboard

- Task of the macro: read the clipboard content of the system clipboard.
- Peculiarities: get access to the system clipboard and check if it is containing text content.
- Possible solution: usage of "`SystemClipboard`" service and "`XTransferable`" interface.

This example shows, how to read a text string from the systems clipboard and write it to a Writer document. First a service called "`SystemClipboard`" is needed, to get access the systems clipboard. Its "`XTransferable`" interface provides access the clipboards content itself as well as the supported content types of the current content. The biggest problem reading the clipboard is to ensure that its content is of a compatible type. In this example a text string shall be copied out of the clipboard, but if the clipboard contains a picture or something different than a text, there is no chance of retrieving a text string. Also a text string may support different formats like ASCII or various Unicode formats. Therefore a list of all supported conversion types of the content must be retrieved and searched for a compatible type. If the expected type is present in this list the clipboard content supports the conversion. The conversion is done by calling "`getTransferData`" delivering the expected type as parameter. This example is searching for an UTF-16 format.

To use this string it needs to be converted from UTF-16 to ASCII format using the "`Converter`" service. The "`XTypeConverter`" interface will be used to access this service. Following these instructions the result is a simple string containing the system clipboards content. Finally the current date is appended to the string and the "`XTextDocument`" and "`XSimpleText`" interface of our currently opened document are used to write this string to the current text cursor position. The current cursor is retrieved using the current controller object of the text document.

e_ReadClipboard.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end
```

```
-- this is the text we want to write
cliptext = ""

-- create the clipboard service controlling the clipboard of
-- the operating system
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
clipboard = "com.sun.star.datatransfer.clipboard.SystemClipboard"
s_Clipboard = x_MultiServiceFactory~createInstance(clipboard)
x_Clipboard = s_Clipboard~XClipboard

-- get the clipboard content
x_Transferable = x_Clipboard~getContents()

/*
check if the clipboard object contains text information:
get a list of all supported data types
and search for a valid entry (utf-16)
*/

flavorslist = x_Transferable~getTransferDataFlavors()
flavorslistlength = bsf('arrayLength', flavorslist)
counter = 1
found = false;
do while (counter <= flavorslistlength) & (found = false)
  found = (flavorslist[counter]~bsf.getFieldValue("MimeType") = "text/plain;charset=utf-16")
  counter = counter + 1
end

-- if it contains a valid entry:
if found then
do
  counter = counter - 1
  -- create a data type converter service
  s_Converter = x_MultiServiceFactory~createInstance("com.sun.star.script.Converter")
  x_TypeConverter = s_Converter~XTypeConverter

  -- transform data (into simple text)
  content = x_Transferable~getTransferData(flavorslist[counter])

  -- read clipboard as string
  stringtype = bsf.getConstant("com.sun.star.uno.TypeClass", "STRING")
  cliptext = x_TypeConverter~convertToSimpleType(content, stringtype)

end

-- add the current date to the previously recieved text
cliptext = cliptext || DATE("E",,,".")

-- and write down this information at the current textcursor position
x_TextDocument = x_Document~XTextDocument
x_Text = x_TextDocument~getText

s_CurrentController = x_TextDocument~getCurrentController()
x_TextViewCursorSupplier = s_CurrentController~XTextViewCursorSupplier
x_CurrentCursor = x_TextViewCursorSupplier~getViewCursor()

x_TextCursor = x_Text~createTextCursorByRange(x_CurrentCursor~getStart())

x_SimpleText = x_Text~XSimpleText
x_SimpleText~insertString(x_TextCursor, cliptext, .false)

::requires UNO.CLS
```

Sourcecode 14: e_ReadClipboard.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO07].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextViewCursorSupplier interface: Cf. [IDLRef41].

Example specific references: Cf. [IDLRef29], [IDLRef30], [IDLRef31], [IDLRef32], [IDLRef33], [IDLRef34], [IDLRef35], [IDLRef42], [IDLRef43].

getConstant(): Cf. [REFBSF03].

Screenshots of the result:

Before invoking the macro copy a text out of a simple text editor, as seen in figure23.

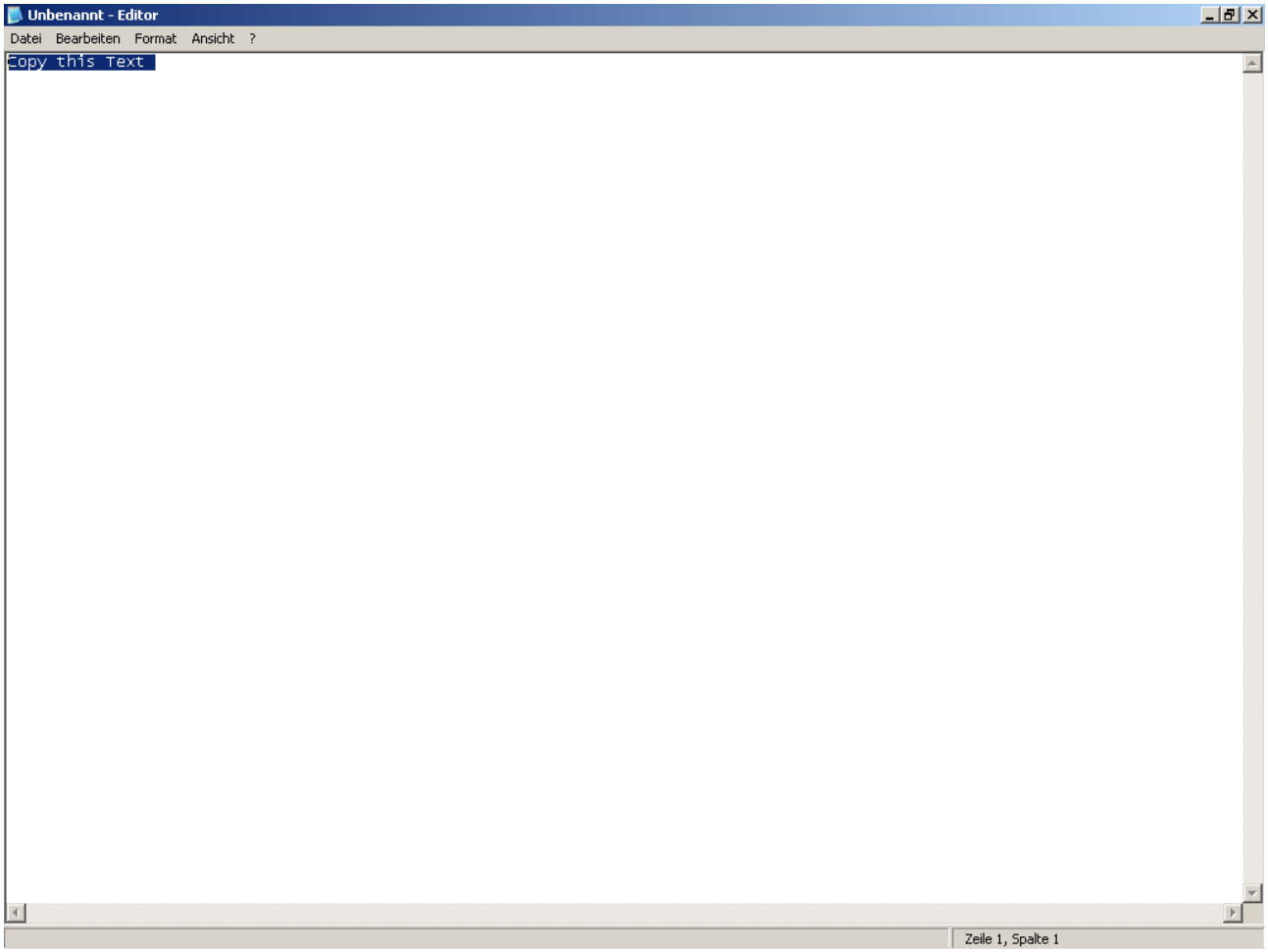

Figure 23: Copy a simple text.

If the macro is executed it is adding some date information to the extracted clipboard string and inserts this string to a writer document. See also figure 24 below.

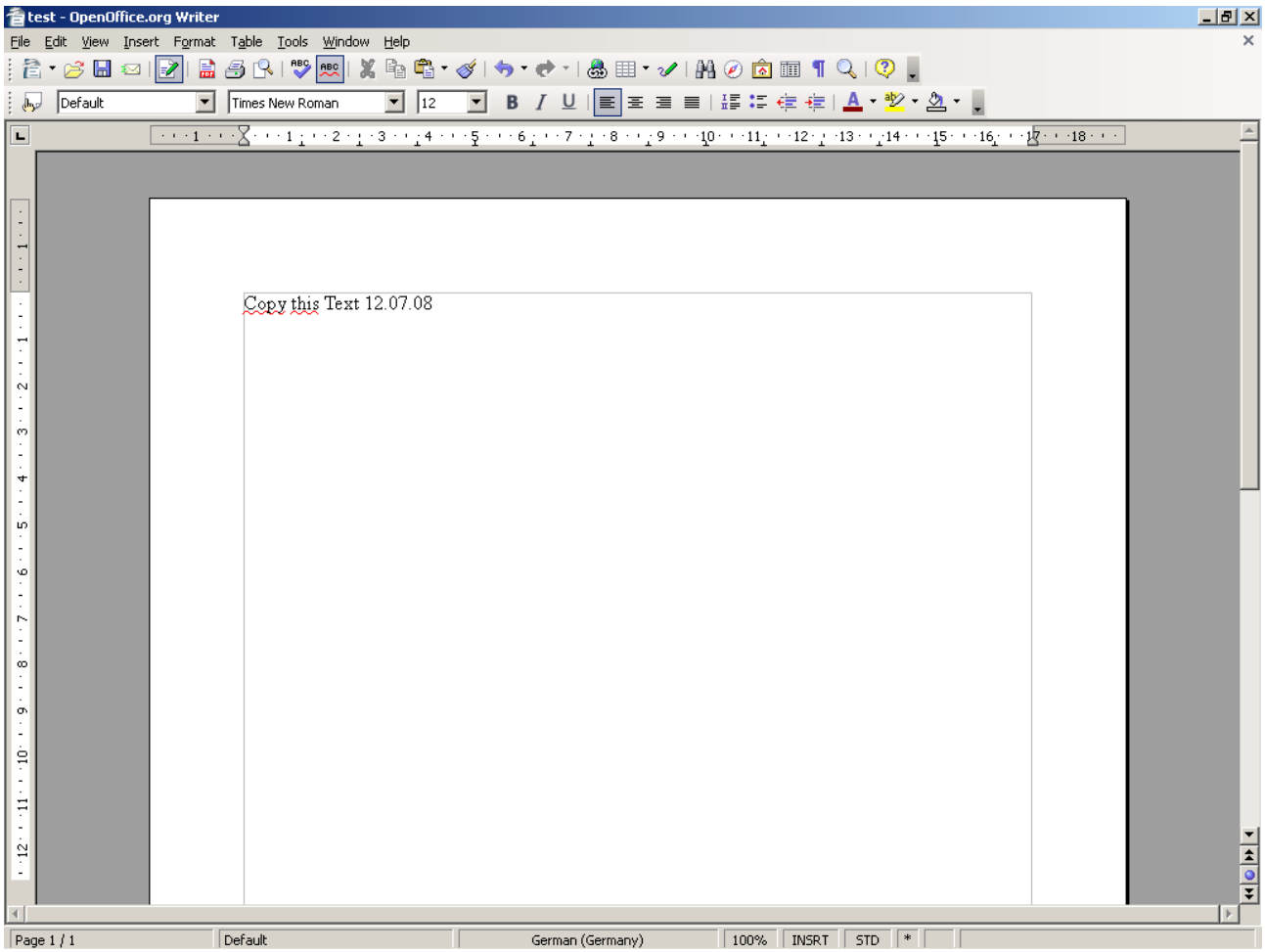

Figure 24: Output of e_ReadClipboard.rex.

6.1.6 Example 6: Read the Version and Language of Open Office

- Task of the macro: read the version and language information of Open Office and view them in a message box.
- Peculiarities: get access to the configuration service.
- Possible solution: usage of "`ConfigurationProvider`" service.

To get the version and language information of Open Office, access to the configuration of Open Office is needed. By instantiating the "`ConfigurationProvider`" service this access is provided. This service implements a factory that allows to create configuration ob-

jects which provide read only access to the Open Office configuration. The configuration consists of a tree structure and its leafs contain the values. When creating a configuration object some information must be added, where the interesting configuration values are located within the tree structure. This information is provided by preparing a "`PropertyValue`" array with a size of 1, setting the name of this property to "`nodepath`" and the value to the path of the configuration item. The returned configuration object is a named container. To get the language setting of Open Office, a path information containing "`/org.openoffice.Setup/L10N`" must be submitted and "`getByName("ooLocale")`" must be called on the container. The version is receivable by submitting "`/org.openoffice.Setup/Product`" as path and reading the "`ooSetupVersion`" entry of the returned container. Finally these informations are put together in a string and displayed to the user in a message box.

e_OOVersionandLocale.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first we create an instance of the configurationprovider
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
configprovider = "com.sun.star.configuration.ConfigurationProvider"
s_ConfigProvider = x_MultiServiceFactory~createInstance(configprovider)
x_ConfigFactory = s_ConfigProvider~XMultiServiceFactory

-- this strig tells the provider to give us reading access to the configuration
conf = "com.sun.star.configuration.ConfigurationAccess"

-- here we define the path to the language information
args = uno.CreateArray(.UNO~PROPERTYVALUE, 1)

args[1] = uno.createProperty("nodepath", "/org.openoffice.Setup/L10N")

-- finally we request configuration access to the predefined configurationpath
s_ConfigurationAccess = x_ConfigFactory~createInstanceWithArguments(conf, args)
x_NameAccess = s_ConfigurationAccess~XNameAccess
```

```
-- and read one of its entries
locale = x_NameAccess~getByName("ooLocale")

-- next we try to read another configuration entry:

args[1]~value = "/org.openoffice.Setup/Product"

s_ConfigurationAccess = x_ConfigFactory~createInstanceWithArguments(conf, args)
x_NameAccess = s_ConfigurationAccess~XNameAccess

version = x_NameAccess~getByName("ooSetupVersion")

-- this is a carriage return line feed string to jump into the next
-- line when displaying our information
crlf = "13"~d2c || "10"~d2c

-- combine outputs
output = "Open Office: " || crlf || "Version: " || version || crlf || "Language: " || locale

-- show message
.bsf.dialog~messageBox(output, "About:", "information")

::requires UNO.CLS
```

Sourcecode 15: e_OOVersionandLocale.rex

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO08].

Example specific references: Cf. [IDLRef44], [IDLRef45].

XNameAccess interface: Cf. [IDLRef46].

PropertyValue array: Cf. [REFOOO03], [REFOOO04], [REFBSF02], [IDLRef12].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

A screenshot of the messagebox containing the information can be seein in figure 25.

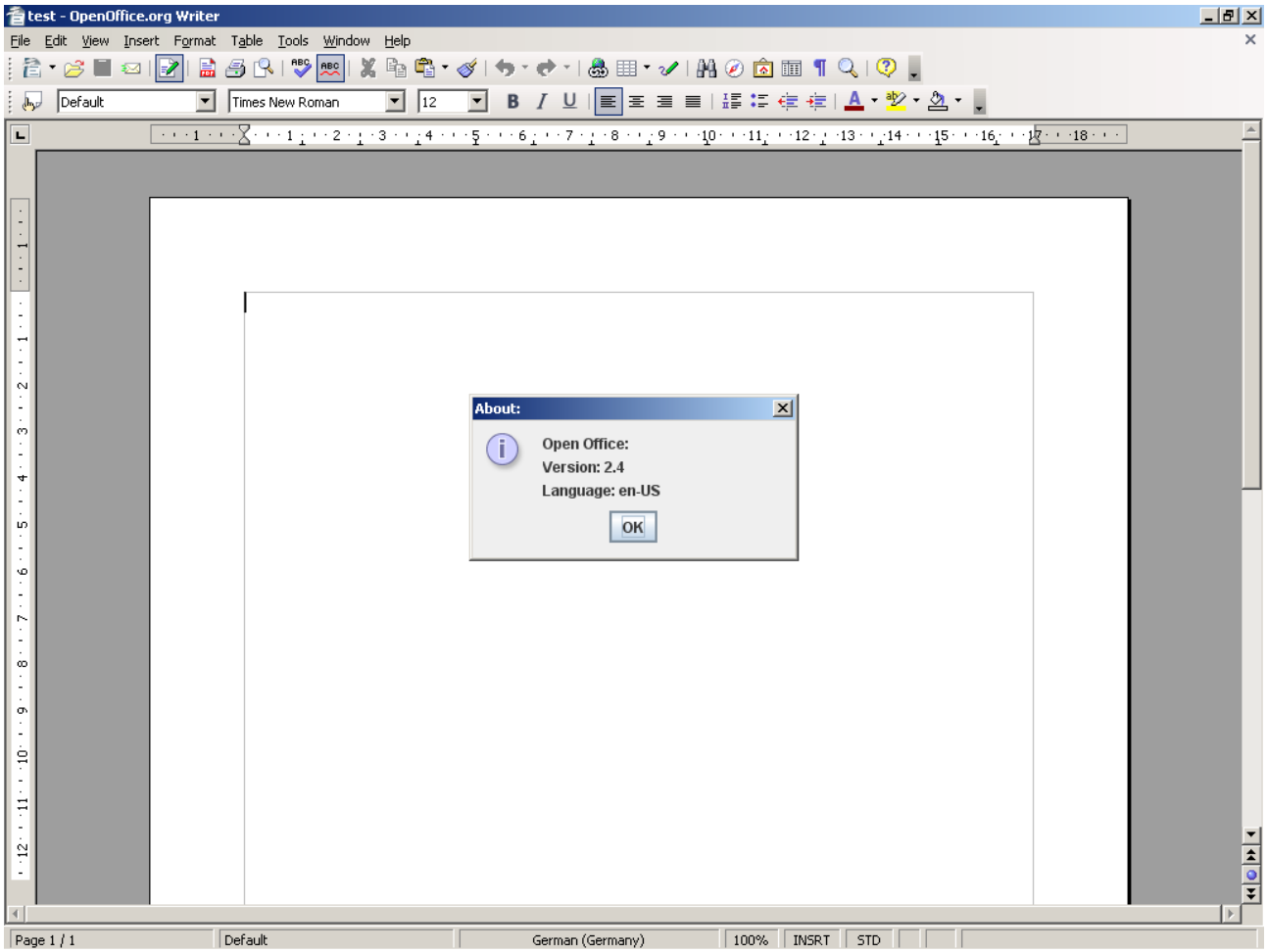

Figure 25: Output of e_OOVersionandLocale.rex.

6.1.7 Example 7: Creating a Dialog with Open Office

- Task of the macro: create a user defined dialog with a button which will change the mouse pointer when it is pressed.
- Peculiarities: creating a dialog, handle its actions[30], get access to the mouse pointer settings.
- Possible solution: usage of control model services for graphical output, event listeners of the "`UNO.CLS`" module to control the actions and the "`Pointer`" service to change the mouse pointer.

[30] Actions like pressing the button or closing the dialog.

This example shows how to create a user defined dialog on runtime. To build such a dialog a window must be created, which is able to contain all the buttons, labels and other visual elements. The dialog window and every element in it, is a UNO service called a Control Model.

These models provide several interfaces to control the appearance and behavior of the window. The "`XPropertyset`" interface provides access to the position and dimension of the window as well as its name, label or tabulator index. To add sub windows to a window the model implements a factory interface to create new windows, as well as a container to register these windows as childes of the parent window. These models also inherit a "`XControlModel`" interface, which provides access to the behavior of the window. The "`XControlModel`" interfaces of the registered sub windows are accessible by the "`XControlContainer`" interface provided by the parent window.

After creating a main dialog window and adding some controls to it, the main windows Window Peer needs to be registered to a Toolkit service. This allows the window to be displayed. Additionally the main window needs to be initialized by a "`Frame`" service, so pressing the close button will effectuate the disposal of the window.

The next step is to constitute the behavior of the controls. In this example pressing the button and closing the dialog will trigger an event. Therefore two Event Listeners must be added to the the Button by calling the "`addEventListener`" method of BSF4Rexx. The first event listener will be called when the button has been pressed[31]. The second event listener will be called when the close button of the dialog has been pressed[32]. The button also gets a dispose event, because, if the main window is closed it sends a disposing command to all its childes. If one of these events is triggered, then the string stated by the third parameter of the "`addEventListener`" method is appended to the event queue of BSF4Rexx. The only thing left to do, to react on these events, is to periodically check the event queue for new messages. The BSF4Rexx command "`pollEventText(1000)`" will wait 1000 milliseconds and will then return the next event message.

If our button is pressed, the shape of the mouse pointer, when it is moving over the button the next time, shall change. To achieve this, a "`Pointer`" service needs to be instantiated. This service is representing a mouse pointer and by applying this service to the "`Windowpeer`" of the button the mouse pointer will change.

[31] Use action identifier string "`ACTIONPERFORMED`" when registering an Eventlistener.
[32] Use action identifier string "`DISPOSING`" when registering an Eventlistener.

To make the dialog visible, refrain from using the dialogs execute method, because the execution of the macro will be halted until the dialog is closed. This would cause the events not being processed until the dialog closes. Instead the usage of the "`XWindow`" interface of the main windows control is recommended to set the dialogs visibility to true.

Another advice is to call the dispose method of the dialogs "`XComponent`" interface at the end of the macro to assure, that the dialog is always terminated.

Currently there are Problems calling this macro inside of Open Office. Doing so might result in Open Office hanging up.

e_Dialog.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



/*
**********************************
* Dialog Window
**********************************
*/

x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory

o_Dialog = x_MultiServiceFactory~createInstance("com.sun.star.awt.UnoControlDialogModel")

o_Dialog.Properties = o_Dialog~XPropertyset

o_Dialog.Properties~setPropertyValue("PositionX", box("int", 64))
o_Dialog.Properties~setPropertyValue("PositionY", box("int", 64))
o_Dialog.Properties~setPropertyValue("Width",     box("int", 256))
o_Dialog.Properties~setPropertyValue("Height",    box("int", (128-8)))
o_Dialog.Properties~setPropertyValue("Title",     "How much do you like ooRexx?")

o_Dialog.Factory = o_Dialog~XMultiServiceFactory
o_Dialog.Container = o_Dialog~XNameContainer

o_Dialog.Control =
x_MultiServiceFactory~createInstance("com.sun.star.awt.UnoControlDialog")~XControl
o_Dialog.Control~setModel(o_Dialog~XControlModel)
o_Dialog.ControlContainer = o_Dialog.Control~XControlContainer
```

```
s_Toolkit = x_MultiServiceFactory~createInstance("com.sun.star.awt.Toolkit")
x_Toolkit = s_Toolkit~XToolkit

o_Dialog.Control~createPeer(x_Toolkit, .nil)

-- needed to let the Dialog dispose, if just displayed but not executed
s_Frame = x_MultiServiceFactory~createInstance("com.sun.star.frame.Frame")
x_Frame = s_Frame~XFrame

x_Frame~initialize(o_Dialog.Control~XWindow)


/*
**********************************
* Label1
**********************************
*/

o_Label1 = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlFixedTextModel")
o_Label1.Properties = o_Label1~XPropertySet

o_Label1.Properties~setPropertyValue("PositionX", box("int", 8))
o_Label1.Properties~setPropertyValue("PositionY", box("int", 40))
o_Label1.Properties~setPropertyValue("Width",     box("int", (256-16)))
o_Label1.Properties~setPropertyValue("Height",    box("int", 32))
o_Label1.Properties~setPropertyValue("Name",      "Label1")
o_Label1.Properties~setPropertyValue("TabIndex",  box("short", 1))
o_Label1.Properties~setPropertyValue("Label",     "Prefix1")

o_Dialog.Container~insertByName("Label1", o_Label1);

/*
**********************************
* Button1
**********************************
*/

o_Button1 = o_Dialog.Factory~createInstance("com.sun.star.awt.UnoControlButtonModel")
o_Button1.Properties = o_Button1~XPropertySet

o_Button1.Properties~setPropertyValue("PositionX", box("int", 8))
o_Button1.Properties~setPropertyValue("PositionY", box("int", 8))
o_Button1.Properties~setPropertyValue("Width",     box("int", (256-16)))
o_Button1.Properties~setPropertyValue("Height",    box("int", 32))
o_Button1.Properties~setPropertyValue("Name",      "Button1")
o_Button1.Properties~setPropertyValue("TabIndex",  box("short", 0))
o_Button1.Properties~setPropertyValue("Label",     "Click Me")

o_Dialog.Container~insertByName("Button1", o_Button1);
o_Button1.Control = o_Dialog.ControlContainer~getControl("Button1")

-- o_Button1.Control~XButton~bsf.addEventListener("action", "",               "noAction")
o_Button1.Control~XButton~bsf.addEventListener("action", "actionPerformed","Action")
o_Button1.Control~XButton~bsf.addEventListener("action", "disposing",      "close")



-- change mouse pointer apperance, whilst moving over label
-- define cursortypes
CursorWait = bsf.getConstant("com.sun.star.awt.SystemPointer", "WAIT")
CursorGo = bsf.getConstant("com.sun.star.awt.SystemPointer", "ARROW")

-- change cursortype
s_Pointer = x_MultiServiceFactory~createInstance("com.sun.star.awt.Pointer")
x_Pointer = s_Pointer~XPointer

cursorstate = 1


-- do not call "o_Dialog.Control~XDialog~execute()" otherwise
-- you do not have instant Listener report
o_Dialog.Control~XWindow~setVisible(.true)

do forever
   eventText=bsf.pollEventText(1000) -- timeout of 100/1000 sec
   if eventText="close" then leave;   -- disposed
   if eventText="Action" then
   do
      if cursorstate then
      do
         x_Pointer~SetType(CursorWait)
```

```
            o_Button1.Control~getPeer()~setPointer(x_Pointer)
            cursorstate = 0
         end
         else
         do
            x_Pointer~SetType(CursorGo)
            o_Button1.Control~getPeer()~setPointer(x_Pointer)
            cursorstate = 1
         end
      end
end

o_Dialog.Control~XComponent~dispose()

::requires UNO.CLS
```

Sourcecode 16: e_Dialog.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Example specific references: Cf. [IDLRef47], [IDLRef48], [IDLRef49], [IDLRef50], [IDLRef51], [IDLRef52], [IDLRef53], [IDLRef54], [IDLRef55], [IDLRef56], [IDLRef57], [IDLRef58], [IDLRef59], [IDLRef60], [IDLRef61], [IDLRef62], [REFBSF04], [REFBSF06], [REFBSF07].

getConstant(): Cf. [REFBSF03].

Visual output of this macro:

Figure 26 shows the Dialog, created by the macro.

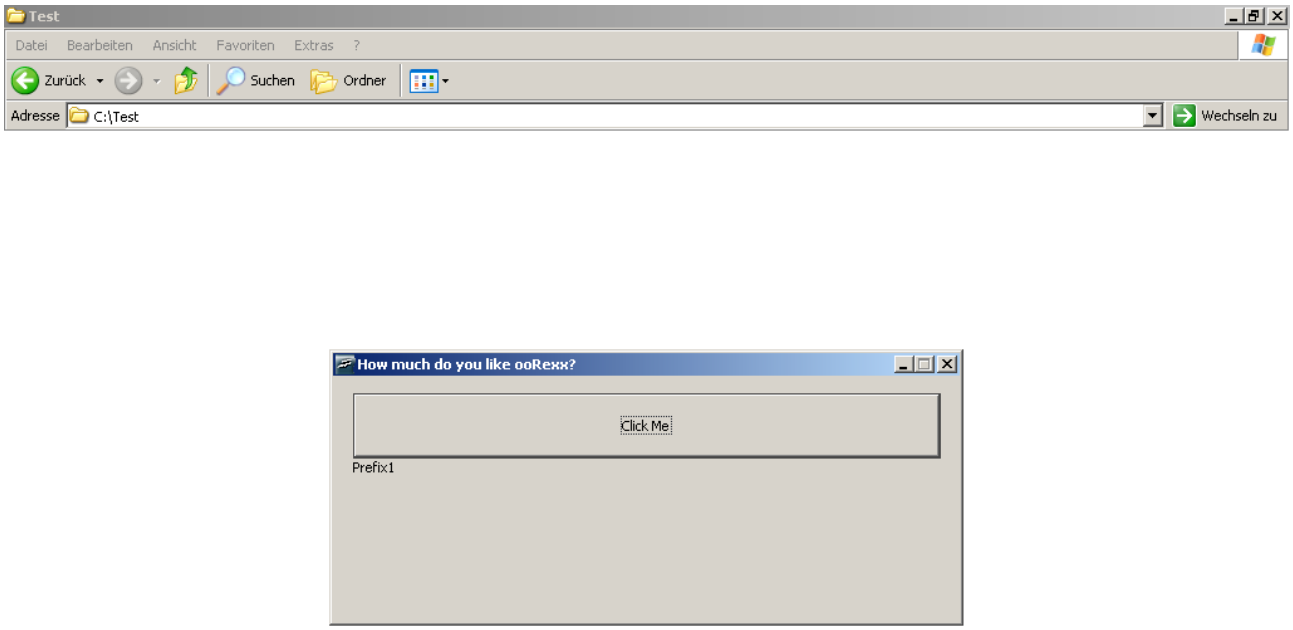

Figure 26: The Dialog.

6.1.8 Example 8: Create an Open Office Addon

- Task of the macro: create an Open Office addon out of a ooRexx macro library.
- Peculiarities: create a "`.zip`" file and get access to its content, as well as create a text file inside the zip file and directly write into it.
- Possible solution: usage of "`Package`" service for the zip file and the "`Pipe`" service for writing the text file.

Creating Open Office addons is quite easy, because they are zip files with a description file called "`manifest.xml`". The Open Office API also provides a service to read and write such addons, named "`Package`". Always keep in mind that every library needs a

description file named "`parcel-descriptor.xml`", so if the chosen directory is not containing such a file, the addon will not be able to function properly. Therefore this example searches the selected path for such a file using the "`SimpleFileAccess`" service, which also provides methods to read and manipulate external files.

After the path to a valid library has been selected, the destination folder is requested from the user and the name of the library followed by a ".oxt" extension is added to the path to describe the destination file. In the next step the "`Package`" service must be created and initialized using the "`XInitialization`" interface. The initialization is required to prepare the zip file and to tell the package to store its content to the destination file by stating the previously defined file path as parameter for the "initialize" method. This parameter must be a string array of size 1. Now the package is ready to be filled.

But before the package is able to incorporate the library files, the package needs a folder representing the library within the package. To do so, the "`XHirarchicalNameAccess`" interface, which allows navigating trough the packages tree like folder structure, must be obtained. Using this interface the root stream object can be retrieved by calling the "`getByHierarchicalName`" method, submitting an empty string as parameter. Next the "`XSingleServiceFactory`" interface of the package needs to be retrieved, to create new file or folder stream objects for our package. Creating a folder object is done by calling the "createInstanceWithArguments" method with a boolean object array as parameter, containing only one boolean object set to true. If the created stream object shall be a file, submit the same array containing a boolean object set to false. To append the newly created folder stream object to the package, the "`insertByName`" method of the root folders "`XNameContainer`" interface must be called, providing the new folder or file name as first parameter and the stream object as second parameter. After the folder has been added to the package, the newly created folder stream object is retrieved by using the "`getByHierarchicalName`" method submitting the name of the folder as parameter. To make this folder a library, the "`XPropertyset`" interface of the folder stream object is needed to set the folders "`MediaType`" property to "`application/vnd.sun.star.framework-script;type=ooRexx`".

Finally all the script files and the parcel descriptor must be copied into the package. To do so the script iterates through a list of all files contained in the library folder. For each file a new file stream object is created and its "`XActiveDataSink`" interface is retrieved. Then

the corresponding external file is opened for reading, and its content copied to the file object by calling the "`XActiveDataSink`s" "`setInputStream`" method which requires the input stream of the opened file as parameter. Also each file must be added to the library folder stream object. To commit all changes done to the real zip file, the "`commitChanges`" method of the packages "`XChangesBatch`" interface is called.

Additionally this example is adding a text file containing the date and time of the export. The biggest problem of this feature is to directly write into a file within the package. To solve this problem a "`Pipe`" service is created which is able to connect the file stream and the text output stream by using a buffer. Because this service is using a buffer, this buffer must be cleared before it is used by calling its "`flush`" method. And after the text has been written to the text stream the "`closeOutput`" method has to be called in order to send a signal to the pipe that the buffer is ready to be copied to the file stream.

This example is using the folder picker dialog, described more closely in the example at 6.1.2, to receive the folder containing the library and the destination folder of the addon. The viewed path of the dialog is set to the path where the user defined libraries are located using the service "`PathSubstitution`" and its "`XStringSubstitution`" interface.

When creating such packages watch out for macros or tool bars calling other macros and don't forget to add these macros too. Also one should watch out for the macro URLs, which will get invalid by exporting, because a library of an addon will be registered as "`<PackageFileName>.<libraryname>`". How to address macros within a package is described in chapter 4.3.

e_ExportLibrary.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
```

```
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first we will ask for the foldername name of the library using the folder dialog
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
folderpicker = "com.sun.star.ui.dialogs.OfficeFolderPicker"
s_FolderDialog = x_MultiServiceFactory~createInstance(folderpicker)
x_FolderDialog = s_FolderDialog~XFolderPicker

-- better name for our dialog:
x_FolderDialog~setDescription("Select User Library to Export")

-- get Path to user Macros
s_pathsubst = x_MultiServiceFactory~createInstance("com.sun.star.util.PathSubstitution")
x_stringsubst = s_pathsubst~XStringSubstitution
usermacropath = x_stringsubst~getSubstituteVariableValue("$(user)") || "/Scripts/oorexx"

-- set Directory of Dialog to user macro directory
x_FolderDialog~setDisplayDirectory(usermacropath)

-- start the folder dialog
pathchoosen = x_FolderDialog~execute()
if ( pathchoosen ) then
do
   -- if ok button pressed:
   -- read selected path
   librarypath = x_FolderDialog~getDirectory()

   -- create file access interface to access files
   s_SimpleFileAccess = x_MultiServiceFactory~createInstance("com.sun.star.ucb.SimpleFileAccess")
   x_SimpleFileAccess = s_SimpleFileAccess~XSimpleFileAccess

   -- check if parcel-descriptorfile can be found in library directory
   islibrary = x_SimpleFileAccess~exists(librarypath || "/parcel-descriptor.xml")

   if (islibrary) then
   do
      -- if parcel deskriptor- file can be found:
      -- use Folder Picker again to get destination folder
      x_FolderDialog~setDescription("Select Export destination")

      if ( x_FolderDialog~execute() ) then
      do
         -- if OK-button pressed:
         -- generate package filename with pathname
         libraryname = getLastFromURL(librarypath)

         savefile = x_FolderDialog~getDirectory() || "/" || libraryname || ".oxt"

         -- if the file already exists, delete it
         delete = SysFileDelete(uno.convertFromURL(savefile))
         if (delete > 2) then
         do
            -- check for file deletion errors (32 = file is blocked by other thread)
            errortext = "File delete error: " || delete || ", accessed by other thread?"
            .bsf.dialog~messageBox(errortext, "ERROR", "error")
            exit 0
         end

         -- now create package object
         s_Package = x_MultiServiceFactory~createInstance("com.sun.star.packages.Package")
         x_PackageInit = s_Package~XInitialization

         -- initialize the package
         c_String = .bsf4rexx~Class.class~forName("java.lang.String")
         initargs = .bsf~bsf.createArray(c_String, 1)
         initargs[1] = savefile
         x_PackageInit~initialize(initargs)

         -- get access to the directory structure of the zip file
         x_HierarchicalNameAccess = s_Package~XHierarchicalNameAccess

         -- get the root item and its containerinterface
         o_RootPackageStream = x_HierarchicalNameAccess~getByHierarchicalName("")
         x_RootNameContainer = o_RootPackageStream~XNameContainer

         -- now create a factory which is able to create new subdirectories and files
```

```
        x_PackageFactory = s_Package~XSingleServiceFactory

        -- arguments to create a directory
        -- arguments MUST BE OBJECTS not primitive types!
        .bsf~bsf.import("java.lang.Boolean", "c_Boolean")
        dirargs = .bsf~bsf.createArray(.c_Boolean, 1)
        dirargs[1] = .c_Boolean~new("true")

        s_PackageFolder = x_PackageFactory~createInstanceWithArguments(dirargs)

        -- insert directory object into package
        x_RootNameContainer~insertByName(libraryname, s_PackageFolder)

        -- go into the new created directory and query a container interface for it
        o_LibraryPackageStream = x_HierarchicalNameAccess~getByHierarchicalName(libraryname)
        x_LibraryNameContainer = o_LibraryPackageStream~XNameContainer

        -- make this directory an ooRexx script library
        x_LibraryPropertySet = o_LibraryPackageStream~XPropertySet
        scripttype = "application/vnd.sun.star.framework-script;type=ooRexx"
        x_LibraryPropertySet~setPropertyValue("MediaType", scripttype)

        -- set arguments to create a file
        fileargs = .bsf~bsf.createArray(.c_Boolean, 1)
        fileargs[1] = .c_Boolean~new("false")

        -- get all files within librarypath
        libraryfiles = x_SimpleFileAccess~getFolderContents(librarypath, 0)

        -- go trough all files and add them to the package directory
        libraryfileslength = libraryfiles~items
        do counter = 1 to libraryfileslength

           -- first create a fileobject and get the datasink interface of it
           o_FilePackageStream = x_PackageFactory~createInstanceWithArguments(fileargs)
           x_ActiveDataSink  = o_FilePackageStream~XActiveDataSink

           -- next open a file and get its inputstream
           x_InputStream = x_SimpleFileAccess~openFileRead(libraryfiles[counter])

           -- tell the datasink where to read the data from
           -- this starts the reading process
           x_ActiveDataSink~SetInputStream(x_InputStream)

           -- now insert the filled file object to the package
           filename = getLastFromURL(libraryfiles[counter])
           x_LibraryNameContainer~insertByName(filename, o_FilePackageStream)

        end

        /*
        Here we write a file directly into the zip file.
        To do so we first create a new subdirectory and enter it.
        Next a pipeobject is created, this allows sending data from
        an outputinterface to an inputinterface. Now we create a
        Textoutputstream and write data in it. Finally we use a
        Datasink object like the one before to read the data and
        store it in the created file.
        */

        s_InfoPackageFolder = x_PackageFactory~createInstanceWithArguments(dirargs)

        x_RootNameContainer~insertByName("PACKAGEINFO", s_InfoPackageFolder)

        s_InfoPipe = x_MultiServiceFactory~createInstance("com.sun.star.io.Pipe")
        x_InfoPipeOutputStream = s_InfoPipe~XOutputStream
        x_InfoPipeInputStream = s_InfoPipe~XInputStream

        textstream = "com.sun.star.io.TextOutputStream"
        s_InfoTextOutputStream = x_MultiServiceFactory~createInstance(textstream)
        x_InfoActiveDataSource = s_InfoTextOutputStream~XActiveDataSource
        x_InfoActiveDataSource~setOutputStream(x_InfoPipeOutputStream)
        x_InfoTextOutputStream = s_InfoTextOutputStream~XTextOutputStream

        -- clear stream
        x_InfoPipeOutputStream~flush()

        crlf = "0d"x || "0a"x
        texttowrite = 'This Package was created by ExportLibrary macro by Josef Frysak' || crlf
        texttowrite = texttowrite || 'on ' || DATE("L") || ' at ' || TIME("C")
        x_InfoTextOutputStream~writeString(texttowrite)
```

```
         -- write into stream
         x_InfoPipeOutputStream~closeOutput()

         o_InfoPackageStream = x_HierarchicalNameAccess~getByHierarchicalName("PACKAGEINFO")
         x_InfoNameContainer = o_InfoPackageStream~XNameContainer

         o_InfoFileStream = x_PackageFactory~createInstanceWithArguments(fileargs)
         x_InfoActiveDataSink = o_InfoFileStream~XActiveDataSink

         x_InfoActiveDataSink~SetInputStream(x_InfoPipeInputStream)
         x_InfoNameContainer~insertByName("info.txt", o_InfoFileStream)

         -- if all changes are done, we write the data to the zipfile
         x_ChangesBatch = s_Package~XChangesBatch
         x_ChangesBatch~commitChanges()

      end

   end
   else
   do
      -- a message in case the selected library is not containing
      -- a parcel-descriptor file
      errortext = "Selected Directory is not a Library: no parcel-descriptor found "
      .bsf.dialog~messageBox(errortext, "ERROR", "error")
   end

end

-- a simple routine to parse the name of the file or the directory name
getLastFromURL:
   use arg url
   return RIGHT(url, LENGTH(url) - LASTPOS("/", url))

::requires UNO.CLS
```

Sourcecode 17: e_ExportLibrary.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO09].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Folder Picker: Cf. [IDLRef22], [IDLRef23].

Example specific references: Cf. [IDLRef48], [IDLRef62], [IDLRef63], [IDLRef64], [IDLRef65], [IDLRef66], [IDLRef67], [IDLRef68], [IDLRef69], [IDLRef70], [IDLRef71], [IDLRef72], [IDLRef73], [IDLRef74], [IDLRef75], [IDLRef76], [IDLRef77], [IDLRef78].

Messagebox: Cf. [REFBSF01].

How this example works out:

First the user must select a library. This is displayed in figure 27.

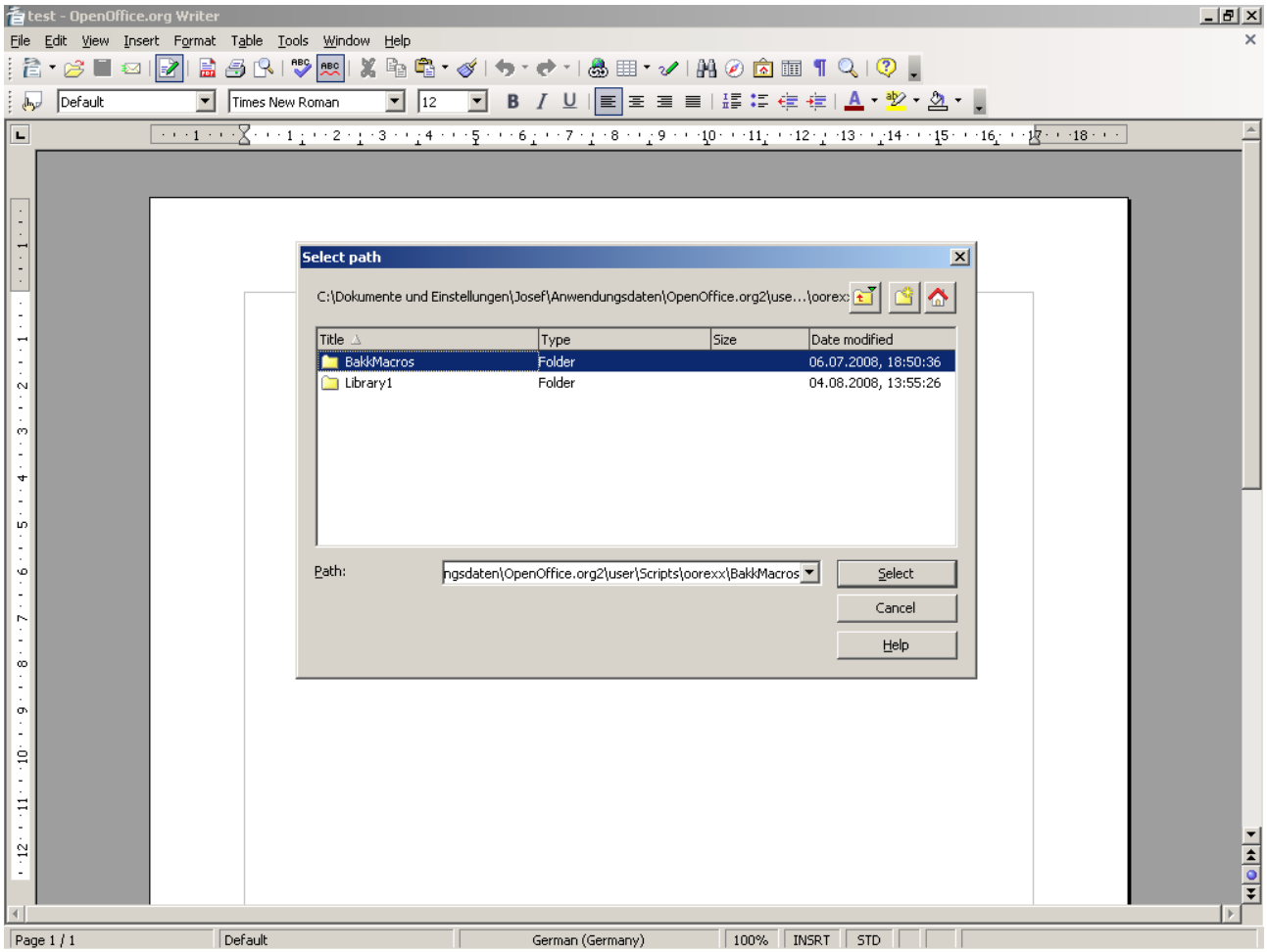

Figure 27: Library folder selection.

Next the user has to select a destination Folder, as seen in figure 28.

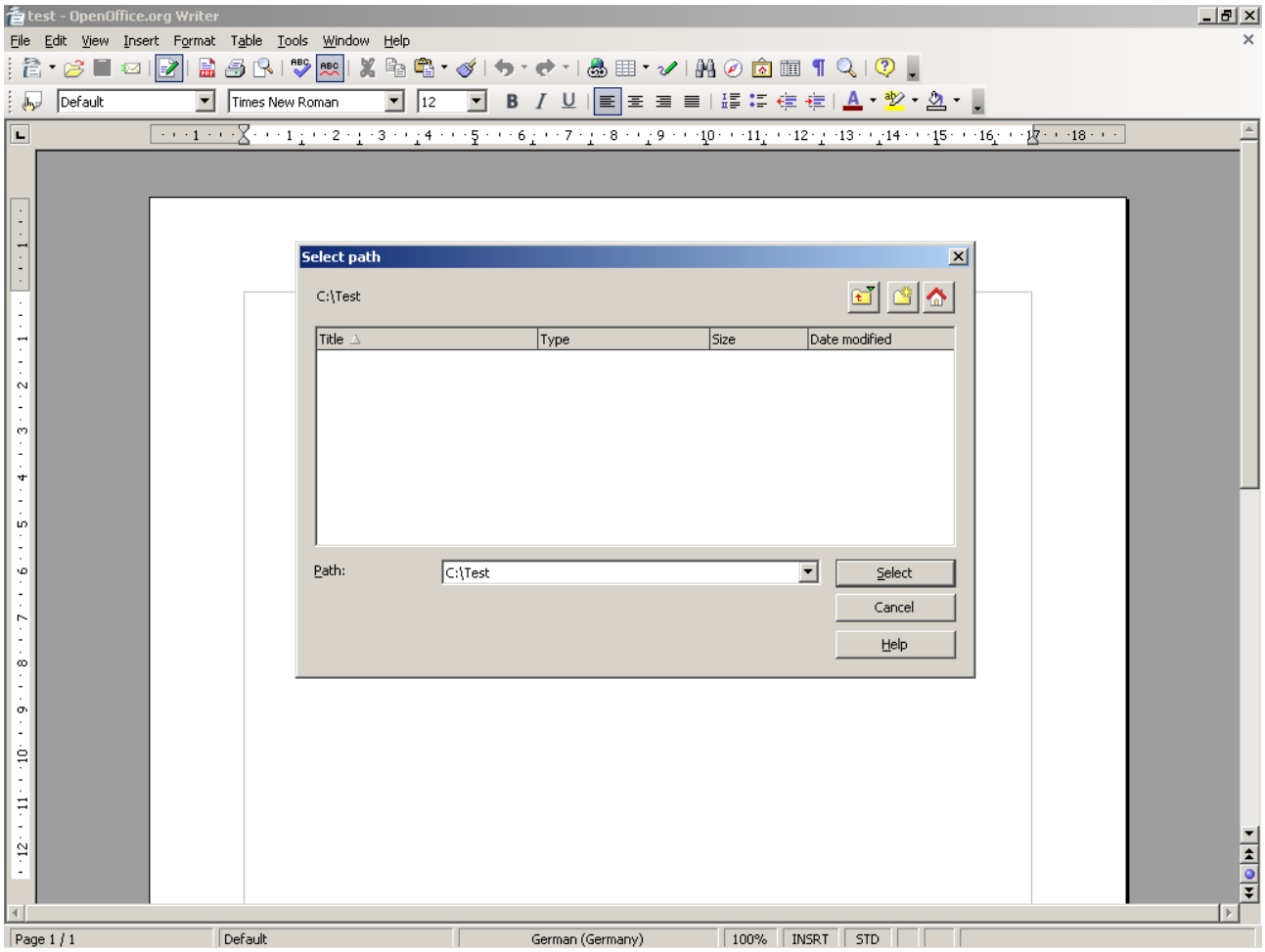

Figure 28: Destination folder selection.

After the selections have been made, the created addon can be found at the specified folder, which is depicted in figure 29 below.

Figure 29: The addon file.

6.2 Writer examples

The next examples show how to remote control the Writer program of Open Office.

6.2.1 Example 1: Create custom paragraph style

- Task of the macro: automatically create a new paragraph style named code.
- Peculiarities: get access to the documents paragraph styles.
- Possible solution: usage of "`StyleFamilies`" service.

This example serves as a preparation for the following example. It creates and registers a custom paragraph style named „`code`". First it checks whether a paragraph style with such

a name already exists or not, by getting a list of all paragraph styles from the current documents "`StyleFamilies`" service using its "`XNameAccess`" interface. If this paragraph style already exists, an error message appears, otherwise the macro will continue. To build a new custom paragraph style a new paragraph object using the documents "`XMultiServiceFactory`" interface must be created. Now access to the properties of this paragraph style is requested by its "`XPropertySet`" interface. The properties are used to change the background color, left and right margin of the paragraph, the border and border distances, as well as its default font. To make changes to the appearance of the border a "`Borderline`" structure is required. Finally the new paragraph object is added to the current documents "`StyleFamilies`" service using its "`XNameAccess`" interface method "`insertByName`". Now the new custom paragraph style is available.

w_CreateStyleCode.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first we search all paragraph styles if there is allready one
-- with name "code"
x_StyleFamiliesSupplier = x_Document~XStyleFamiliesSupplier
x_StyleFamilies = x_StyleFamiliesSupplier~getStyleFamilies()
s_StyleFamily = x_StyleFamilies~getByName("ParagraphStyles")

x_NameAccess = s_StyleFamily~XNameAccess

if x_NameAccess~hasByName("code") then
do
   -- if "code" allready exists show error message
   .bsf.dialog~messageBox('PageStyle "code" allready exists!', "ERROR", "error")
end
else
do
   -- if "code" does not exist:

   -- create a factory to create a new paragraph style object
   x_MultiServiceFactory = x_Document~XMultiServiceFactory
   s_ParagraphStyle = x_MultiServiceFactory~createInstance("com.sun.star.style.ParagraphStyle")
```

```
   -- now set its properties
   s_ParagraphProperties = s_ParagraphStyle~XPropertySet

   -- WARNING! Java.Integer = UNO_LONG !!!!!!!
   -- its background color
   paracolor = .bsf~new("java.lang.Integer", X2D("FFFFCC"))
   s_ParagraphProperties~setPropertyValue("ParaBackColor", paracolor)

   -- left and right margin
   pramargin = .bsf~new("java.lang.Integer", 500)
   s_ParagraphProperties~setPropertyValue("ParaLeftMargin", pramargin)
   s_ParagraphProperties~setPropertyValue("ParaRightMargin", pramargin)

   -- left, right, top and bottom border
   o_Border = .bsf~new("com.sun.star.table.BorderLine")
   o_Border~bsf.setFieldValue("Color", 0)
   o_Border~bsf.setFieldValue("InnerLineWidth", 0)
   o_Border~bsf.setFieldValue("OuterLineWidth", 1)
   o_Border~bsf.setFieldValue("LineDistance", 0)

   s_ParagraphProperties~setPropertyValue("LeftBorder", o_Border)
   s_ParagraphProperties~setPropertyValue("RightBorder", o_Border)
   s_ParagraphProperties~setPropertyValue("TopBorder", o_Border)
   s_ParagraphProperties~setPropertyValue("BottomBorder", o_Border)

   borderdist = .bsf~new("java.lang.Integer", 50)
   s_ParagraphProperties~setPropertyValue("LeftBorderDistance", borderdist)
   s_ParagraphProperties~setPropertyValue("RightBorderDistance", borderdist)
   s_ParagraphProperties~setPropertyValue("TopBorderDistance", borderdist)
   s_ParagraphProperties~setPropertyValue("BottomBorderDistance", borderdist)

   -- and the fontname
   s_ParagraphProperties~setPropertyValue("CharFontName", "Arial")

   -- finally insert the new paragraph style to the list of
   -- paragraph styles of this document
   x_NameAccess~insertByName("code", s_ParagraphStyle)
end

::requires UNO.CLS
```

Sourcecode 18: w_CreateStyleCode.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO10].

Propertyset: Cf. [IDLRef20], [IDLRef21].

XNameAccess interface: Cf. [IDLRef46].

XStyleFamiliesSupplier interface: Cf. [IDLRef79].

ParagraphStyle: Cf. [IDLRef80].

Example specific references: Cf. [IDLRef81].

setFieldValue(): Cf. [REFBSF05].

Messagebox: Cf. [REFBSF01].

The goal of this macro:

The result of the macro is a newly created paragraph style, like displayed in figure 30.

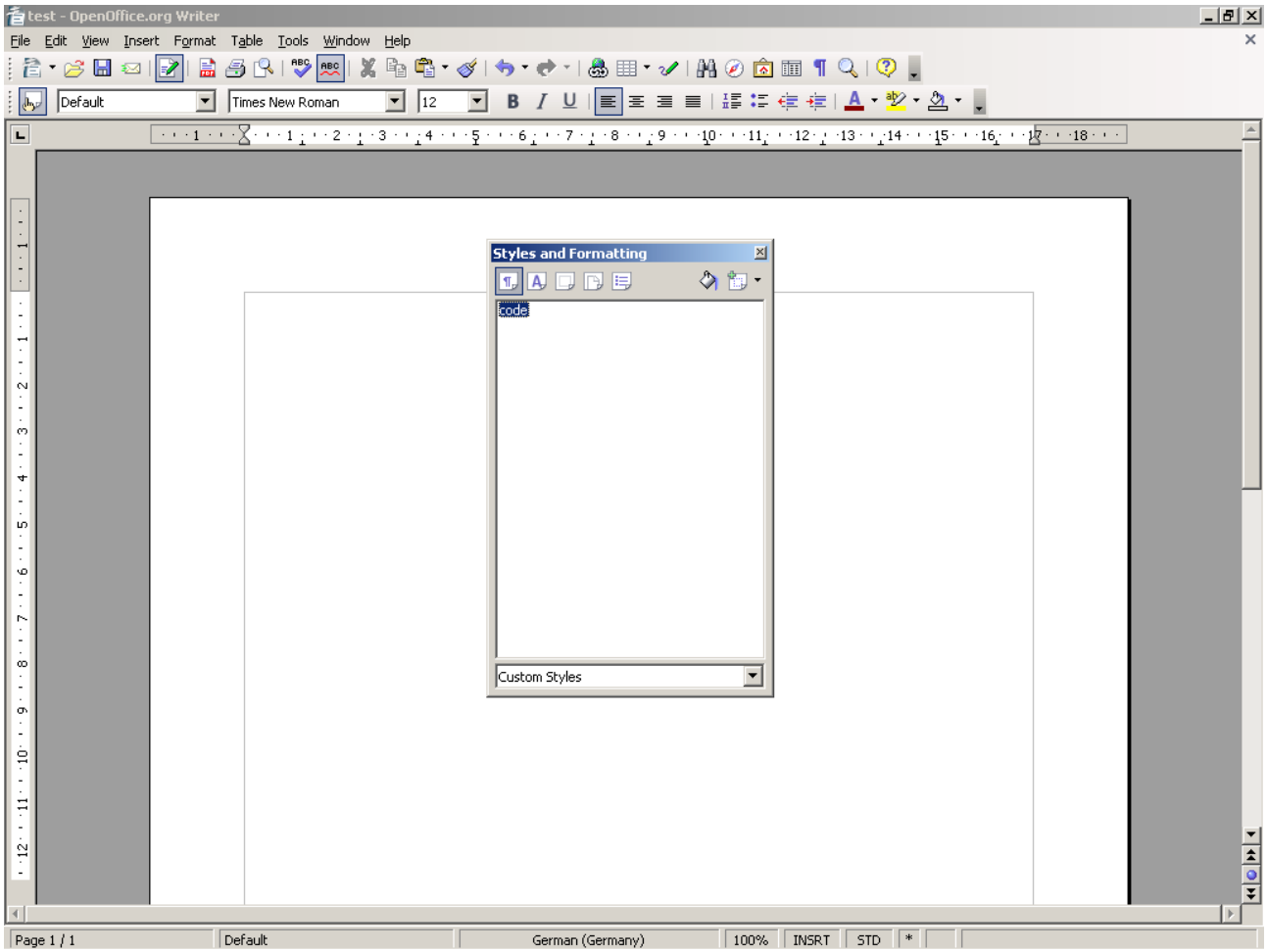

Figure 30: The created paragraph style.

6.2.2 Example 2: Import Code with gvim

- Task of the macro: import a selected text file containing source code and format code text with gvim.
- Peculiarities: call gvim out of an ooRexx script and import a HTML file[33] into a writer document.
- Possible solution: usage of ooRexx capability to execute shell commands and the "`XDocumentInsertable`" interface of the document service to insert the HTML file.

[33] gvim converts the code text to HTML format.

This example is able to import text files containing code of various programming languages. Which languages are supported and how the code is highlighted and formatted depends on the the settings of the external program gvim. Because this example is using this external program, the path to the executable needs to be appended to the systems path environment variable, so the macro is always able to find gvim. Also this example uses a paragraph style named "`code`", which must be defined before starting this macro. The paragraph style is responsible for the background color, the border of the code paragraph, and so on. It helps keeping all code paragraphs consistent. To automatically create a new paragraph style named "`code`" use Example 6.2.1.

When this macro has been started, it first checks if a custom paragraph style named "`code`" exists. If it does not exist, an error message occurs, otherwise the macro continues asking the user for the location of the file containing the code using a file picker dialog.

This file picker dialog is created by using the "`XMultiServiceFactory`" interface of the component contexts service manager. The file dialogs title is changed using the dialogs "`setTitle`" method. Its "`setMultiSelectionMode`" method allows the user to only select one file at the same time, by stating a zero value (equals a boolean false value) as a parameter of this method. The file picker dialog also inherits a "`XFilterManager`" interface which makes it possible to add different kinds of file extensions. To activate the dialog its execute method must be called. If a file has been successfully selected, the execute method returns true, otherwise a boolean value of false is returned. Finally the selected files can be retrieved from the dialog as a string array by calling its "`getFiles`" method. Also read example 6.1.2 for information regarding the problems of such dialogs.

If a file has been selected, gvim is forced to convert the previously selected code file to a HTML format. gvim will store the HTML content to a file using the original filename, including its extension, and appending a ".html" extension to it. In order to locate the HTML file the same procedure has to be done in the script too. Next access to the writer document ("`XTextDocument`", "`XText`" interfaces) and its current cursor position ("`XTextViewCursorSupplier`" interface) is requested. Now the old paragraph style, where the text cursor is currently positioned at, is stored and changed to the custom paragraph style named "`code`". Next the HTML file is inserted at the current cursor position using the "`XDocumentInsertable`" interface. The interfaces method "`insertDocumentFromURL`" requires two parameters. The first one is a URL string containing the filename. The second

one is a "`PropertyValue`" array containing only one entry, which states the filter needed to convert the inserted file format. In this example we need a HTML converter. Finally a new paragraph is created, and the style of the new paragraph is set to the previously stored old paragraph style.

w_ImportCode.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first check if paragraph style "code" is present
x_StyleFamiliesSupplier = x_Document~XStyleFamiliesSupplier
x_StyleFamilies = x_StyleFamiliesSupplier~getStyleFamilies()
s_StyleFamily = x_StyleFamilies~getByName("ParagraphStyles")
x_NameAccess = s_StyleFamily~XNameAccess

if \ x_NameAccess~hasByName("code") then
do
   -- if style is not present
   .bsf.dialog~messageBox('Paragraph Style "code" does not exist!', "ERROR", "error")
end
else
do
   -- if style is present:
   -- ask for filename using the file dialog
   x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
   s_FileDialog = x_MultiServiceFactory~createInstance("com.sun.star.ui.dialogs.OfficeFilePicker")
   x_FileDialog = s_FileDialog~XFilePicker
   x_FileDialogFilters = s_FileDialog~XFilterManager

   -- adding some file extensions to the dialog
   x_FileDialogFilters~appendFilter("OORexx *.rex", "*.rex")
   x_FileDialogFilters~appendFilter("C++ *.cpp", "*.cpp")
   x_FileDialogFilters~appendFilter("All *.*", "*.*")

   -- Better name for our dialog:
   x_FileDialog~setTitle("Select Code File")

   -- selecting more than one file at once dissalowed (to allow it use 1)
   x_FileDialog~setMultiSelectionMode(0)

   -- run dialog
   filechoosen = x_FileDialog~execute()
   if ( filechoosen ) then
   do
      -- if dialog button ok pressed read filename
      file = x_FileDialog~getFiles()
```

```
      codefileurl = file[1]
      codefile = uno.convertFromUrl(codefileurl)

      -- set up gvim command
      command = 'gvim ' || codefile || ' -n -c "runtime! syntax/2html.vim" -c wq -c q'

      -- find out which operating system is present and set up the shell execution
      /* Normally ooRexx is able to determine the operating system itself. But calling
         the macro with rexxj or inside Open Office disables this ability.*/
      if .uno~path.separator=";" then
      do
         -- Windows
         ADDRESS CMD
      end
      else
      do
         -- Linux
         shell=value("SHELL",,"ENVIRONMENT") -- get type of shell
         shell=substr(shell, shell~lastpos("/")+1) -- get shell name
         ADDRESS VALUE shell           -- set shell as command shell
      end

      -- execute command
      command

      htmlfileurl = codefileurl || ".html"

      -- import html file at cursor postion
      x_TextDocument = x_Document~XTextDocument
      x_Text = x_TextDocument~getText

      s_CurrentController = x_TextDocument~getCurrentController()
      x_TextViewCursorSupplier = s_CurrentController~XTextViewCursorSupplier
      x_CurrentCursor = x_TextViewCursorSupplier~getViewCursor()

      x_TextCursor = x_Text~createTextCursorByRange(x_CurrentCursor~getStart())

      -- html insertion interface
      x_DocumentInsertable = x_TextCursor~XDocumentInsertable

      -- current cursor position (and paragraph style)
      x_CursorPropertySet = x_TextCursor~XPropertySet
      oldstyle = x_CursorPropertySet~getPropertyValue("ParaStyleName")

      -- change pararaph style
      x_CursorPropertySet~setPropertyValue("ParaStyleName", "code")

      -- now insert the html file

      /*
      Create a property array.
      Parameter 2 means it has a capacity of two elements
      (if parameter is set to 0 it is not .nil!, but an empty java array).
      */

      properties = uno.CreateArray(.UNO~PROPERTYVALUE, 1)

      properties[1] = uno.createProperty("FilterName", "HTML (StarWriter)")

      x_DocumentInsertable~insertDocumentFromURL(htmlfileurl, properties)

      -- create new paragraph and return to old style
      x_Text~insertControlCharacter(x_TextCursor, 5, .false)
      x_CursorPropertySet~setPropertyValue("ParaStyleName", oldstyle)
   end
end

::requires UNO.CLS
```

Sourcecode 19: w_ImportCode.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO11]

Propertyset: Cf. [IDLRef20], [IDLRef21].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextViewCursorSupplier interface: Cf. [IDLRef41].

XNameAccess interface: Cf. [IDLRef46].

XStyleFamiliesSupplier interface: Cf. [IDLRef79].

ParagraphStyle: Cf. [IDLRef80].

File Picker dialog: Cf. [IDLRef82], [IDLRef83], [IDLRef84], [UMIFOO25].

Example specific references: Cf. [IDLRef85], [REFOOO09].

PropertyValue array: Cf. [REFOOO03], [REFOOO04], [REFBSF02], [IDLRef12].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

First a "`.rex`" file must be selected to let the macro know which file to import. This is shown by figure 31.

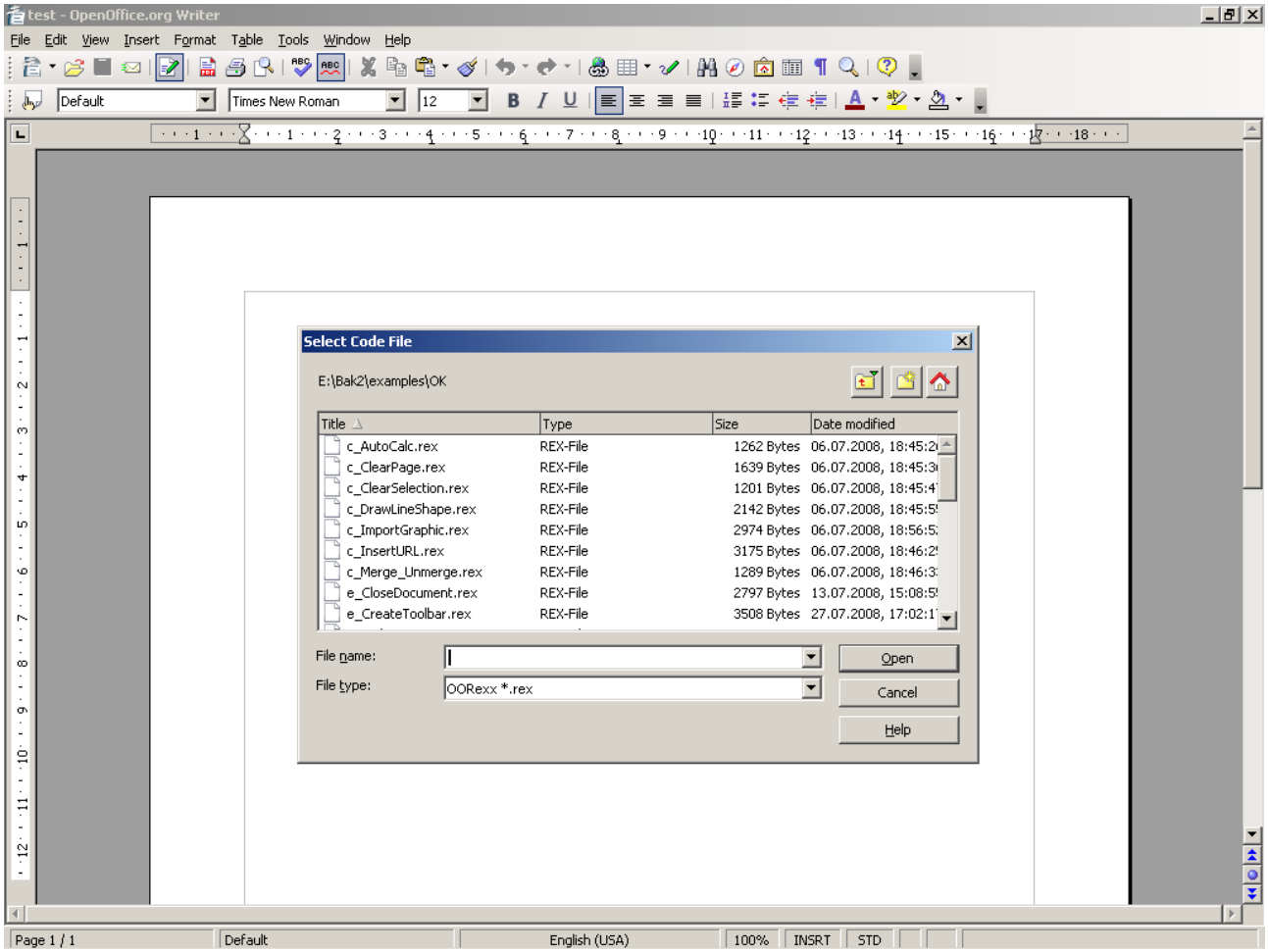

Figure 31: Code file selection.

In this paper this macro has been used to import all the examples and the imports have also been formatted. Therefore figure 32 shows the result of the macro, if no changes have been made to the importet code.

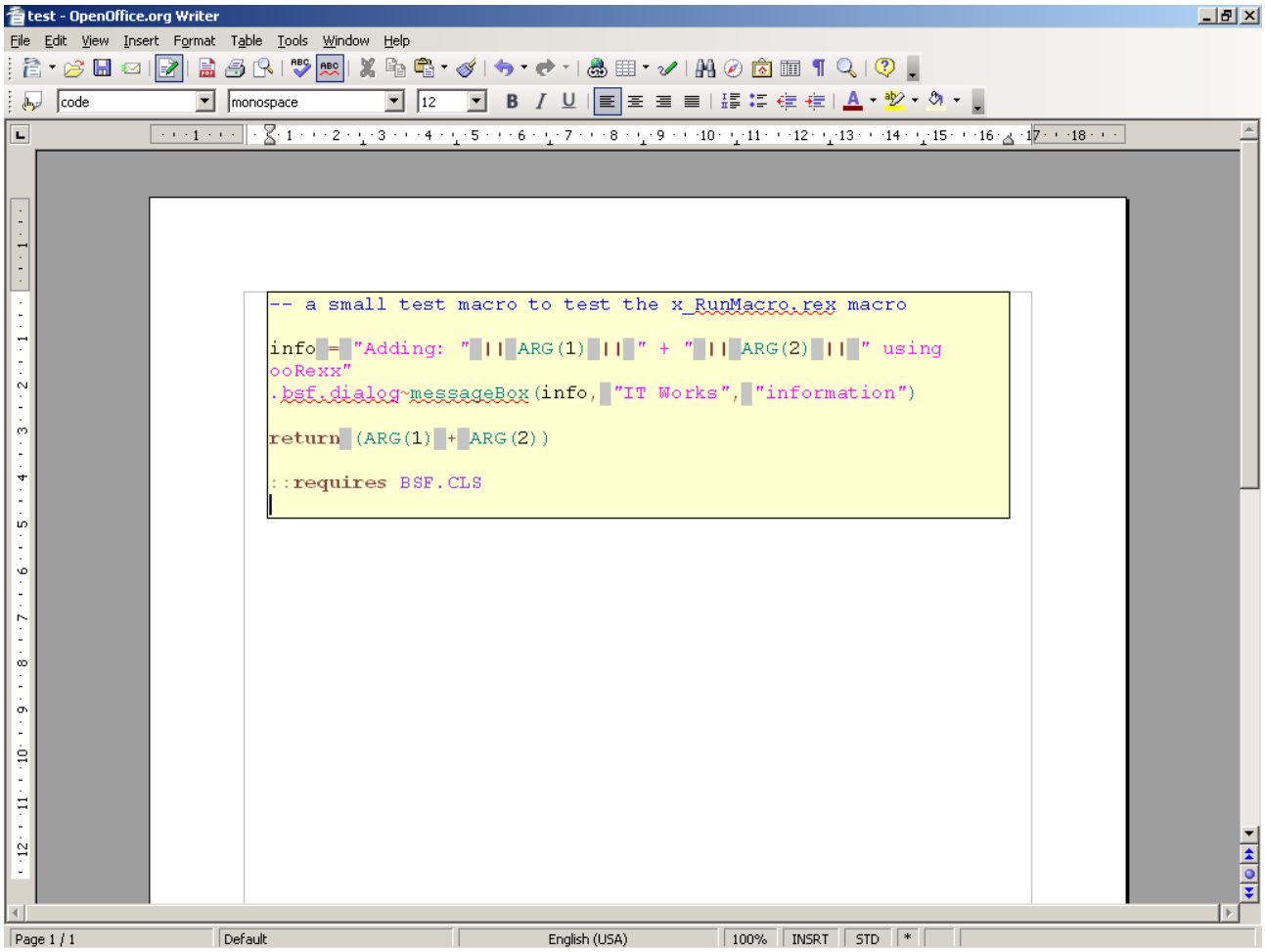

Figure 32: x_Sample.rex imported.

The gray blocks are just specal space characters used by HTML, which can easily be replaced by using search and replace of Open Office.

There are also problems when deleting existing imported code and trying to import code at the very same position. Some colors[34] will depart from the HTML file. Saving the document before reimporting the code helped avoiding this problem.

6.2.3 Example 3: Insert a Date Field

- Task of the macro: insert a date field at the current text cursor position.
- Peculiarities: different date formats for different languages.

[34] Testing the macro, the black colored parts changed to purple.

- Possible solution: usage of "`Locale`" structure to define the language and the "`NumberFormats`" service, which stores, beside other formats, all kinds of date formats.

This Example is inserting a date field at the current cursor position. To achieve this, the text interface and the current cursor interface of the text document must be retrieved. Then the documents "`XMultiServiceFactory`" interface is used to create a new "`DateTime`" text field. Using the "`XPropertyset`" interface of the new text field enables the potential to set the fixed field and the date field attributes of the text field. The next step is to create or retrieve a special number format to display the current date. Due these number formats are queried by a language depended string, a "`Locale`" structure must be instantiated to define the language of the number format string when querying it. Now the "`NumberFormats`" service of the Writer document is used to query the number format submitted as a string by the first parameter and the locale structure as second parameter. If the required number format does not exist, the result of this method will be -1, otherwise it is zero or above. If this index is negative the number format has to be added to the number formats list. The method "`addNew`" will return the index of the appended number format, therefore there should be a positive number format index in either case. Now the "`NumberFormat`" property of the date fields "`XPropertyset`" interface is set to the previously gained index. Finally the documents text service method "`insertTextContent`" is called to add the new date field to the document at the current cursor position.

w_Date.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end
```

```
-- create interface for text document
x_TextDocument = x_Document~XTextDocument
-- get a text object of text document
x_Text = x_TextDocument~getText

-- get the current cursor position
s_CurrentController = x_TextDocument~getCurrentController()
x_TextViewCursorSupplier = s_CurrentController~XTextViewCursorSupplier
x_CurrentCursor = x_TextViewCursorSupplier~getViewCursor()

-- create a textcursor of it
x_TextCursor = x_Text~createTextCursorByRange(x_CurrentCursor~getStart())

-- now create a factory that is able to add datetime objects to the document
x_MultiServiceFactory = x_Document~XMultiServiceFactory
s_DateTime = x_MultiServiceFactory~createInstance("com.sun.star.text.TextField.DateTime")

-- next configure the new date time object
x_DateTimeProperties = s_DateTime~XPropertySet
x_DateTimeProperties~setPropertyValue("IsFixed", .bsf~new("java.lang.Boolean", .true))
x_DateTimeProperties~setPropertyValue("IsDate", .bsf~new("java.lang.Boolean", .true))

-- next create a number formats suppliere to get
-- access to the various automatic number formats
x_NumberFormatsSupplier = x_Document~XNumberFormatsSupplier
x_NumberFormats = x_NumberFormatsSupplier~getNumberFormats()

-- define the language settings to use to search for entries
st_Locale = .bsf~new("com.sun.star.lang.Locale")
st_Locale~bsf.setFieldValue("Language", "en")
st_Locale~bsf.setFieldValue("Country", "USA")

-- look out for the existance of our number format, if none exists, create one.
format ="DD.MM.YY"
formatindex = x_NumberFormats~queryKey(format, st_Locale, .true)
If formatindex = -1 then formatindex = x_NumberFormats~addNew(format, st_Locale)


-- now set the format of the date time object
val = .bsf~new("java.lang.Integer", formatindex)

x_DateTimeProperties~setPropertyValue("NumberFormat", val)

-- finally add the date time object to the text
x_TextContent = s_DateTime~XTextContent
x_Text~insertTextContent(x_TextCursor, x_TextContent, .true)

::requires UNO.CLS
```

Sourcecode 20: w_Date.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO12].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextViewCursorSupplier interface: Cf. [IDLRef41].

Example specific references: Cf. [IDLRef86], [IDLRef87], [IDLRef88].

setFieldValue(): Cf. [REFBSF05].

XTextContent interface: [IDLRef89].

The output of this macro:

The next screenshot, shown in figure 33, shows a date field created by the macr described before.

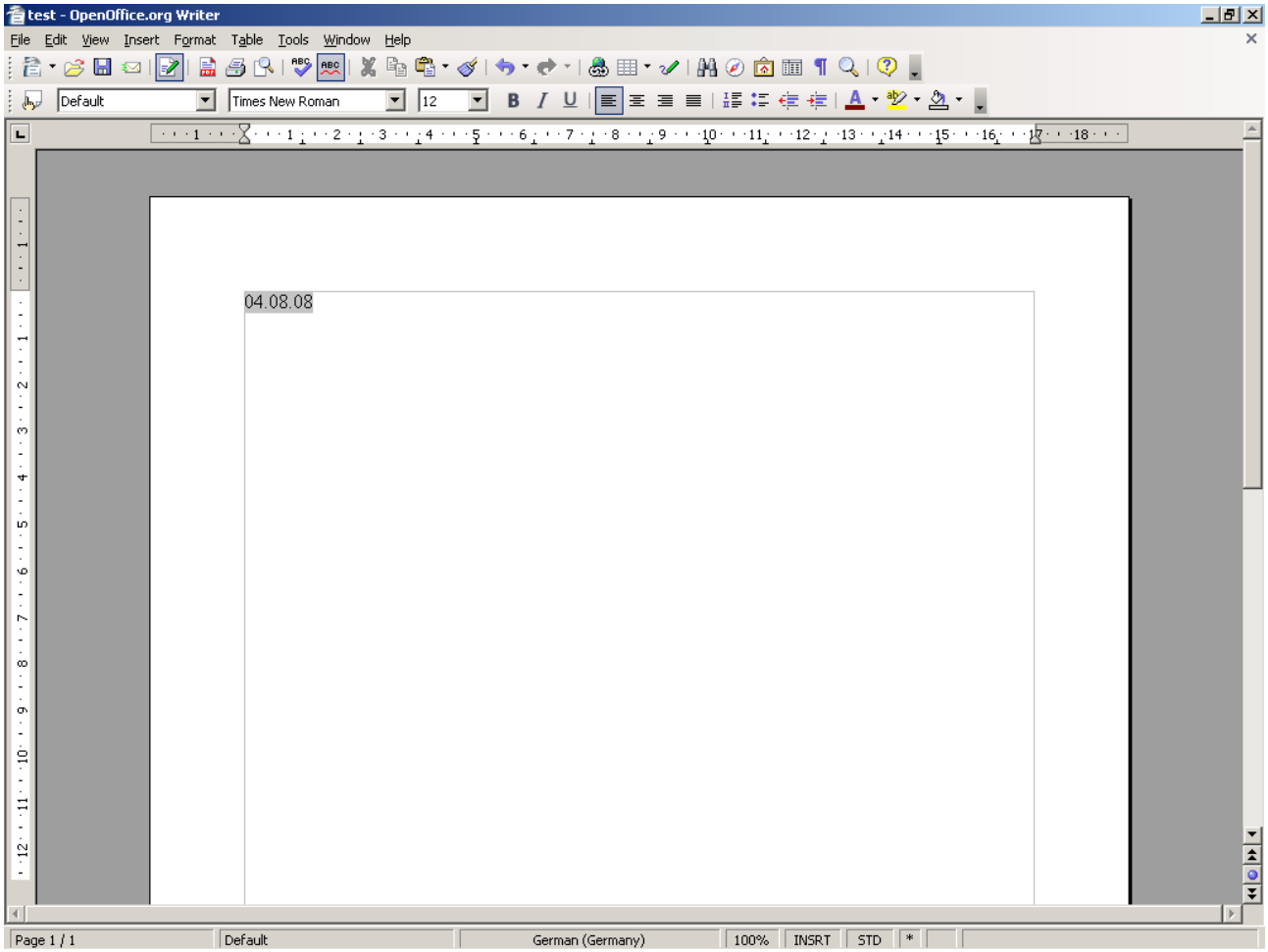

Figure 33: Output of w_Date.rex.

6.2.4 Example 4: Set the Text Locale

- Task of the macro: set the language of all paragraphs to unknown, so the language check ignores them.
- Peculiarities: iterate trough all paragraphs.

- Possible solution: an "`XParagraphCursor`" interface of a virtual text cursor[35].

This example sets the language of the whole text of the Writer document to unknown. The approach is quite easy. First a locale structure is created and its „`Language`" and „`Country`" values are set to a string containing the word „`unknown`". Next the Writer documents text service is gained and used to create a virtual text cursor. The next step is to retrieve the "`XParagraphCursor`" interface of the virtual text cursor. This interface allows jumping from one paragraph to another and to commit changes to the whole paragraph. Now the "`TextCursor`" interface is used to go to the beginning of the text and iterate trough all paragraphs changing each paragraphs "`CharLocale`" property. This will change the locale property of all the paragraphs, but not the content after the last paragraph. Therefore it is necessary to check if there is a content after the last paragraph and if this is the case, then this section must be selected and the selections "`CharLocale`" property modified too.

w_DeactivateTextLocale.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end




-- first we define a locale entry that is unknown
st_Locale = .bsf~new("com.sun.star.lang.Locale")
st_Locale~bsf.setFieldValue("Language", "unknown")
st_Locale~bsf.setFieldValue("Country", "unknown")

-- next we get the text interface of the document
x_TextDocument = x_Document~XTextDocument
x_Text = x_TextDocument~getText

-- now we create a virtual text cursor and a paragraphcursor interface of it.
x_TextCursor = x_Text~createTextCursor()
```

[35] A text cursor created by the macro, which only exists while the macro is executed. It is different from the text cursor the user utilizes.

```
x_ParagraphCursor = x_TextCursor~XParagraphCursor

-- here we go to the start of the document
x_TextCursor~gotoStart(.false)

-- now we jump from one paragraph to the next one and change its locale entry
do while x_ParagraphCursor~gotoNextParagraph(.true)
   x_PropertySet = x_TextCursor~XPropertySet
   x_PropertySet~setPropertyValue("CharLocale", st_Locale)
   x_ParagraphCursor~goRight(0, .false)
end

-- finally we check if there is also text behind the last paragraph.
-- their language also must be changed
if x_ParagraphCursor~gotoEndOfParagraph(.false) then
do
   x_ParagraphCursor~goRight(1, .false)
   x_ParagraphCursor~gotoEnd(.true)
   x_PropertySet = x_TextCursor~XPropertySet
   x_PropertySet~setPropertyValue("CharLocale", st_Locale)
   x_ParagraphCursor~goRight(0, .false)
end

::requires UNO.CLS
```

Sourcecode 21: w_DeactivateTextLocale.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO13].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Example specific references: Cf. [IDLRef88], [IDLRef90].

setFieldValue(): Cf. [REFBSF05].

The effect of this macro:

Figure 34 shows the state of a Writer document before using this macro,

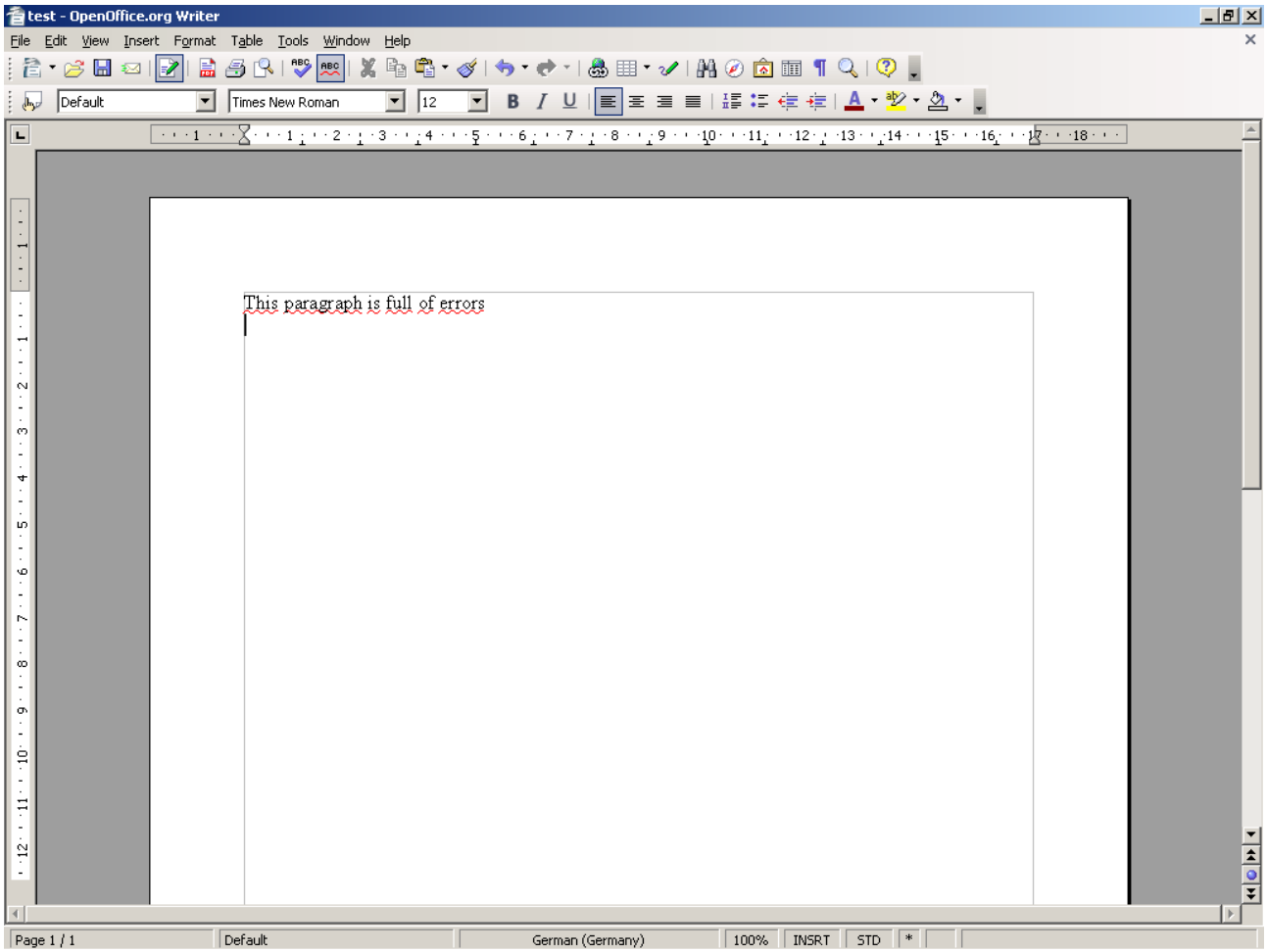

Figure 34: Paragraph with language check enabled.

and figure 35 after applying the macro.

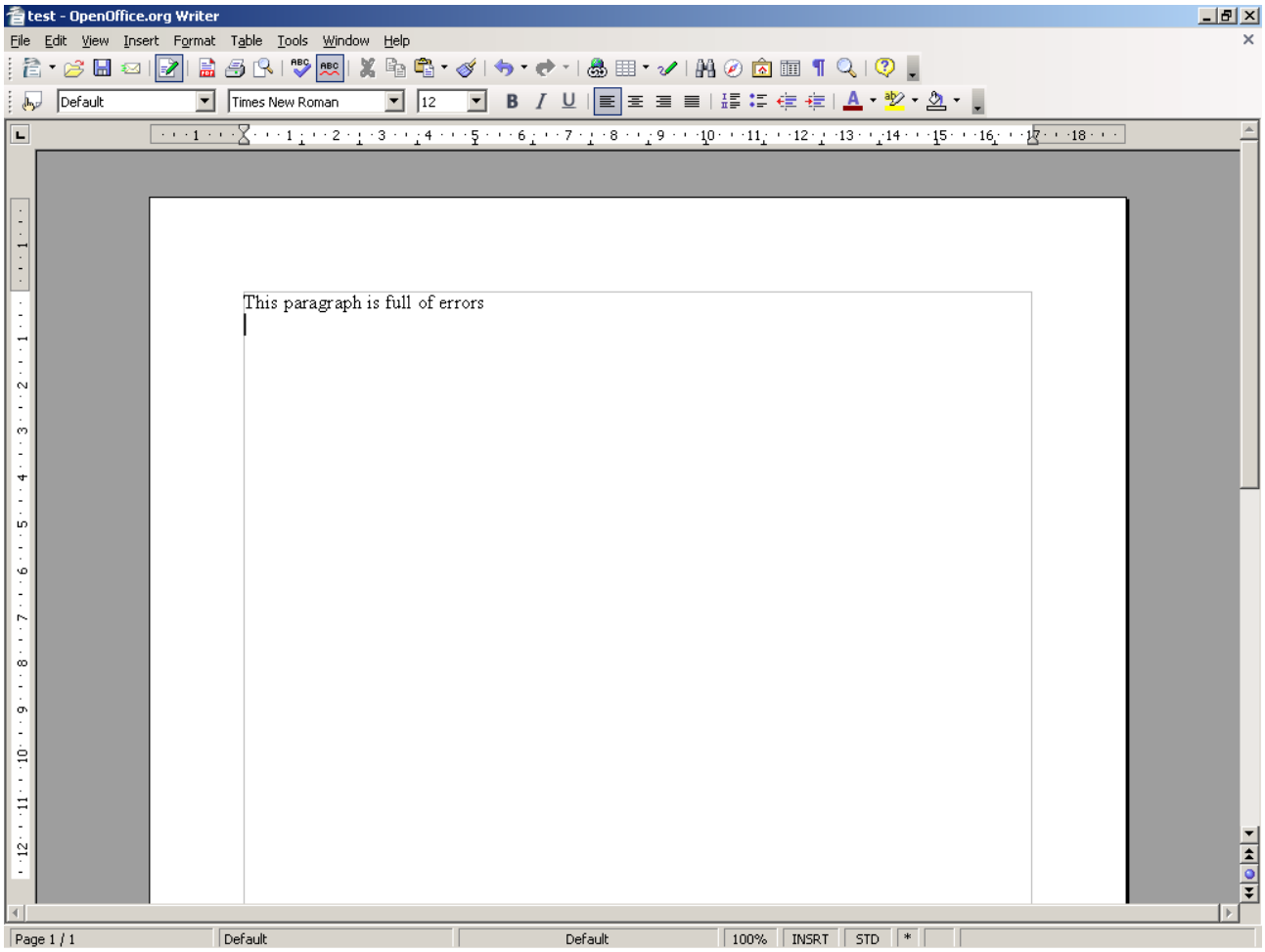

Figure 35: Paragraph with language check disabled.

6.2.5 Example 5: Export all Graphics

- Task of the macro: export all the graphics within a text document to a folder.
- Peculiarities: get access to the pictures of a Writer document.
- Possible solution: usage of the "`XTextGraphicObjectsSupplier`" interface of the document service.

This macro provides the ability to export all pictures of a Writer document to a user defined path. To let the user select the path this example uses the folder picker dialog of Open Office. This dialog is described more closely in the example at 6.1.2. If a folder was selected, the macro starts its task by querying the "`XTextGraphicObjectsSupplier`" which provides access to the container holding all the text graphic objects of a Writer doc-

ument. Calling the "`getGraphicObjects`" function of this interface a container object is returned. This container is accessed by a "`XIndexAccess`" interface to address its elements by a number instead of their names. Next a "`GraphicProvider`" service is created which provides the ability to save pictures to a file. Its method "`storeGraphic`" requires a graphical object as first parameter and a "`PropertyValue`" array to define the graphics format and the URL of the destination file as second parameter. To retrieve the graphical objects the script iterates trough the list of text graphic objects and extracts the graphical objects of the text graphic objects that hold them. This is done by reading out the text graphic objects "`Graphic`" property. Finally the macro stores each retrieved graphical object to a file named "`Picture`" appending its index number.

w_ExportGraphics.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end




-- first we create a folder dialog to select the export folder
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
folderpicker = "com.sun.star.ui.dialogs.OfficeFolderPicker"
s_FolderDialog = x_MultiServiceFactory~createInstance(folderpicker)
x_FolderDialog = s_FolderDialog~XFolderPicker

-- change the displayed text
x_FolderDialog~setDescription("Select Directory to Export Graphics")

pathchoosen = x_FolderDialog~execute()
if ( pathchoosen ) then
do
   -- if a path was sucessfully choosen read the path
   exportpath = x_FolderDialog~getDirectory()

   /*
   now get a text graphics Supplier interface to access all
   the graphics within our text. The function returns a name access
   container but we will use an indexed access.
   */
   x_TextGraphicObjectsSupplier = x_Document~XTextGraphicObjectsSupplier
   x_NameAccess = x_TextGraphicObjectsSupplier~getGraphicObjects()
```

```
   x_IndexAccess = x_NameAccess~XIndexAccess

   -- next we create a Graphicprovider service (to write graphic files)
   graphicprovider = "com.sun.star.graphic.GraphicProvider"
   s_GraphicProvider = x_MultiServiceFactory~createInstance(graphicprovider)
   x_GraphicProvider = s_GraphicProvider~XGraphicProvider

   -- now we iterate trough the text graphic supplier and export all graphics
   arr = uno.CreateArray(.UNO~PROPERTYVALUE, 2)

   amount = x_IndexAccess~getCount()

   do counter = 1 to amount

      -- here we define the type of graphics and the storage path and filename
      arr[1] = uno.createProperty("MimeType", "image/jpeg")
      pictureurl = exportpath || "/" || "picture" || counter || ".jpg"
      arr[2] = uno.createProperty("URL", pictureurl)

      -- retrieve the picture object
      o_Picture = x_IndexAccess~getByIndex(counter - 1)

      -- next read the picture data and get a XGraphic interface of it
      x_TextGrapicPropertySet = o_Picture~XPropertySet
      o_Graphic = x_TextGrapicPropertySet~getPropertyValue("Graphic")
      x_Graphic = o_Graphic~XGraphic

      -- finally store the picture
      x_GraphicProvider~storeGraphic(x_Graphic, arr)
   end
end

::requires UNO.CLS
```

Sourcecode 22: w_ExportGraphics.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO14].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Folder Picker: Cf. [IDLRef22], [IDLRef23].

XIndexAccess interface: Cf. [IDLRef92].

Example specific references: Cf. [IDLRef91], [IDLRef93], [IDLRef94], [IDLRef95].

PropertyValue array: Cf. [REFOOO03], [REFOOO04], [REFBSF02], [IDLRef12].

Screenshots of the output of this macro:

Figure 36 shows a Writer document containing three Pictures and the path selection Dialog of the macro to select the destination folder.

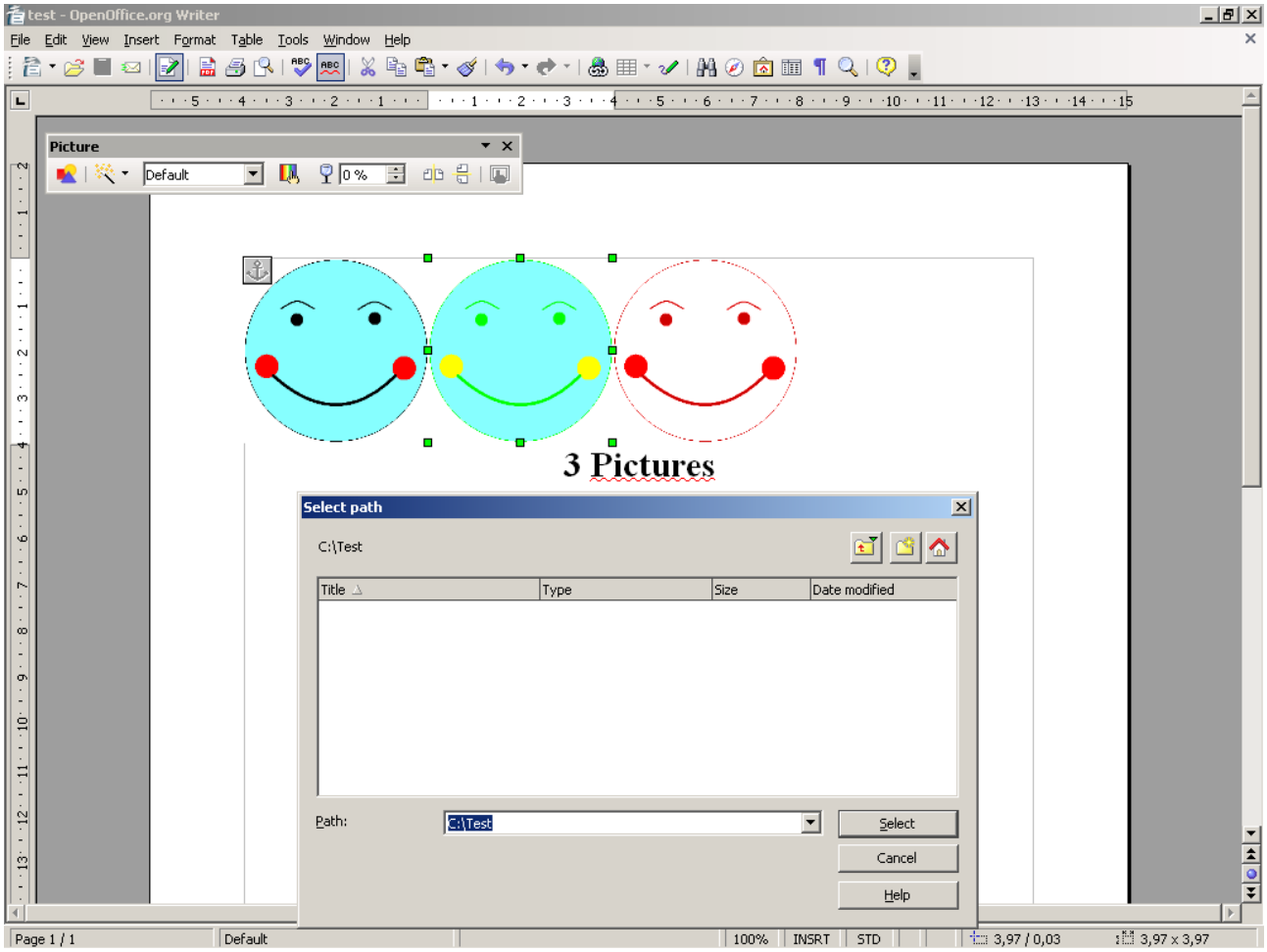

Figure 36: Writer document with three pictures and folder selection dialog of macro.

After running the macro, the exported pictures are located in the previously selected folder, as displayed in figure 37.

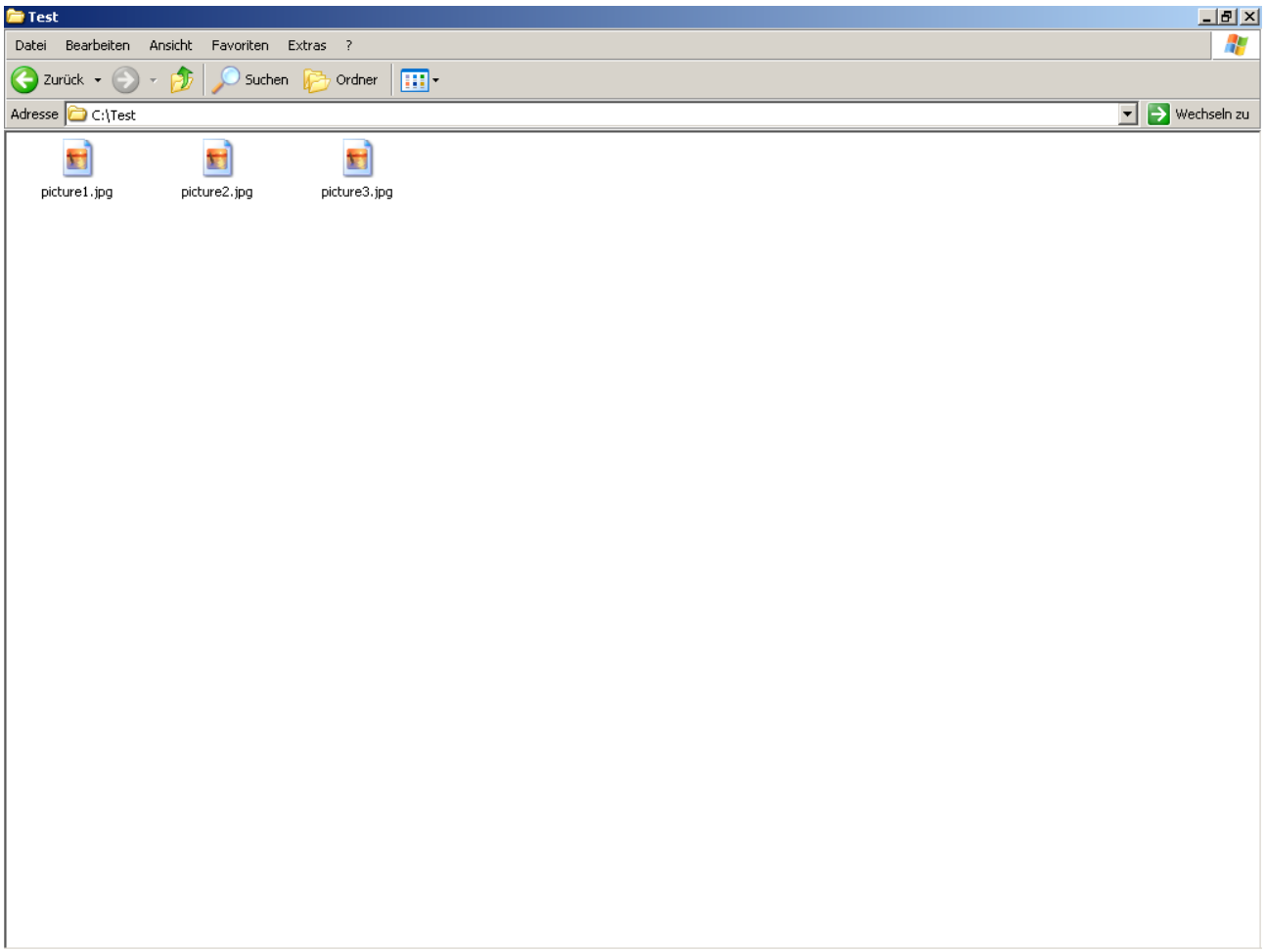

Figure 37: Exported pictures in selected path.

6.2.6 Example 6: Access the Current Selection

- Task of the macro: get the selected text and concatenate it.
- Peculiarities: iterate trough all selection segments.
- Possible solution: usage of the "`getCurrentSelection`" method of the documents "`XModel`" interface to get the selected text segments. Usage of the selections "`XIndexAccess`" interface to iterate trough the selected text segments.

This example is reading the current text selections and combines them to a single string which is displayed by a message box. In a Writer document there may be more than one selections at once. Therefore these selections are stored in a container. To get this con-

tainer the "`getCurrentSelection`" method of the document "`XModel`" interface is called. If no container is returned by this method no text is selected. Iterating trough the container using a "`XIndexAccess`" interface the selections are concatenated together. Finally the result sting is displayed by a message box.

w_GetSelection.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end




-- final output
output = "Text: "

-- get the current selection
x_Model = x_Document~XModel
s_Container = x_Model~getCurrentSelection()

if s_Container <> .nil then
do
   -- if there is a selection iterate trough the selection
   -- and read out the text into output variable

   x_IndexAccess = s_Container~XIndexAccess
   size = x_IndexAccess~getCount()

   do counter = 1 to size
      s_text = x_IndexAccess~getByIndex(counter - 1)
      x_TextRange = s_text~XTextRange

      output = output || x_TextRange~getString()
   end
end

-- finally show what is selected
.bsf.dialog~messageBox(output, "Currently Selected Text:", "information")

::requires UNO.CLS
```

Sourcecode 23: w_GetSelection.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO15].

XModel interface: Cf. [IDLRef17].

XIndexAccess interface: Cf. [IDLRef92].

XTextRange interface: Cf. [IDLRef96].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

Figure 38 shows how the selected text parts are put togetere an a new sentence is returned in a messagebox.

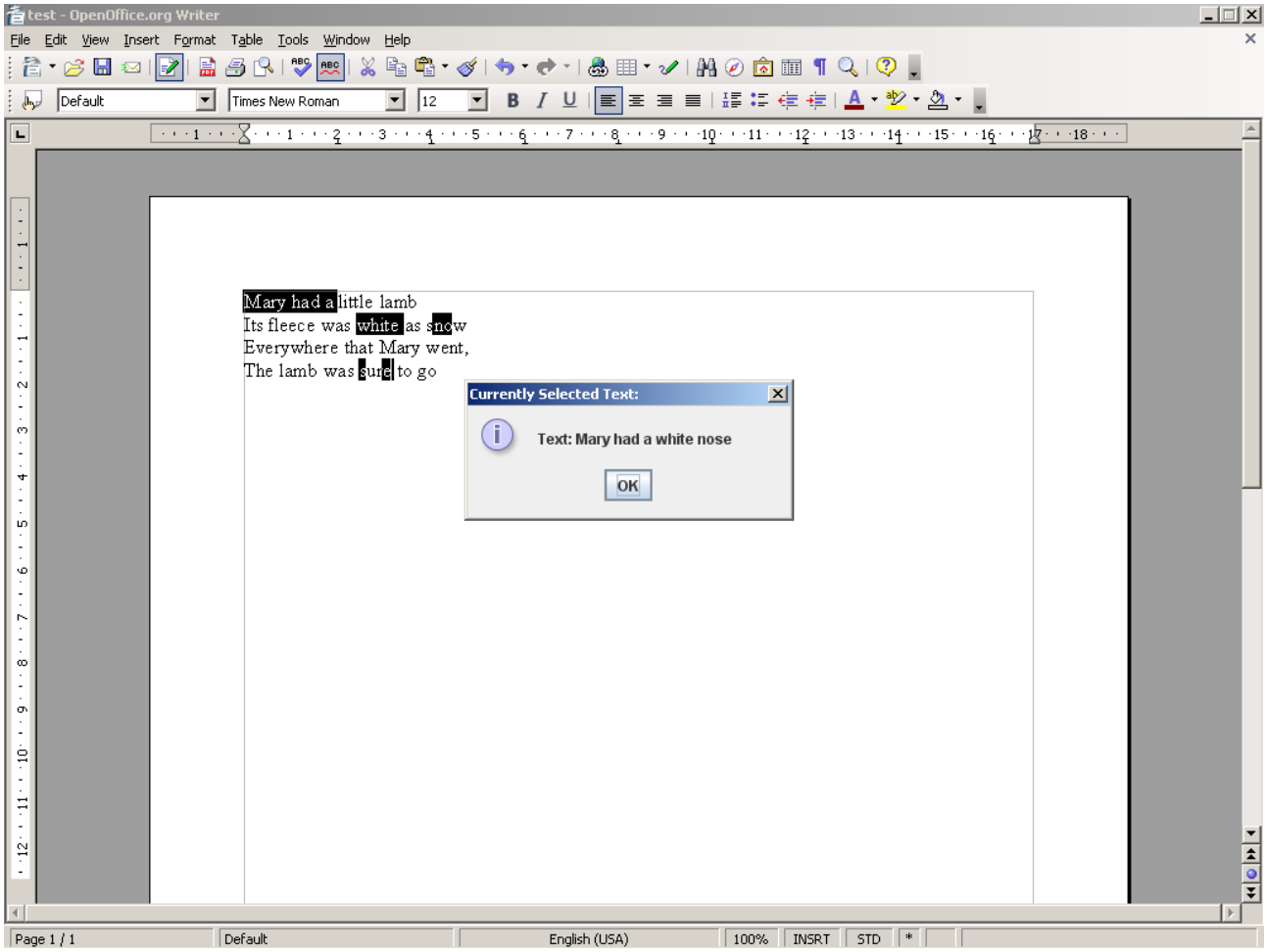

Figure 38: Example of w_GetSelection.rex.

6.2.7 Example 7: Activate and Deactivate Header and Footer

- Task of the macro: activate or deactivate the header and footer of a page.
- Peculiarities: get access to header and footer settings.
- Possible solution: usage of the "`PageStyleName`" property of the current cursor to determine the current page settings and the documents "`StyleFamily`" service to change the header and footer settings.

This example activates and deactivates the header and the footer of the current page type. To do this, first access to the documents text and view cursor interface must be

gained. Then the page style name of the current page must be queried by reading the "`PageStyleName`" property of the current cursor. With this information the corresponding page style object is requested from the documents "`StyleFamily`" service. The service will return a "`PropertySet`" containing all the information of the current pages style. Now the page styles Header and Footer properties are checked if they are set to true, if that is the case they are set to false otherwise they are set to true.

w_Header_Footer.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first we get access to the textdocument and its text object
x_TextDocument = x_Document~XTextDocument
x_Text = x_TextDocument~getText

-- get the current current cursor
s_CurrentController = x_TextDocument~getCurrentController()
x_TextViewCursorSupplier = s_CurrentController~XTextViewCursorSupplier
x_CurrentCursor = x_TextViewCursorSupplier~getViewCursor()

-- get access to the properties of the current cursor and retrieve
-- name of the current pagestyle
cursorproperties = x_CurrentCursor~XPropertySet
pagestylename = cursorproperties~getPropertyValue("PageStyleName")

-- next we search the StyleFamily entries for our current pagestyle
x_StyleFamiliesSupplier = x_Document~XStyleFamiliesSupplier
x_StyleFamilies = x_StyleFamiliesSupplier~getStyleFamilies()
s_StyleFamily = x_StyleFamilies~getByName("PageStyles")
x_NameAccess = s_StyleFamily~XNameAccess
s_PageProperties = x_NameAccess~getByName(pagestylename)

-- get the properties of the current page
pageproperties = s_PageProperties~XPropertySet

-- get the current status of header and footer of this page
oldheader = pageproperties~getPropertyValue("HeaderIsOn")
oldfooter = pageproperties~getPropertyValue("FooterIsOn")

-- if header is on turn it of and vice versa
-- i dont know why disabling needs a complet object as parameter, but
-- enabling does not. (enabling wont work if using objects)
```

```
if oldheader then pageproperties~setPropertyValue("HeaderIsOn", .bsf~new("java.lang.Boolean", 0))
else pageproperties~setPropertyValue("HeaderIsOn", 1)

if oldfooter then pageproperties~setPropertyValue("FooterIsOn", .bsf~new("java.lang.Boolean", 0))
else pageproperties~setPropertyValue("FooterIsOn", 1)

::requires UNO.CLS
```

Sourcecode 24: w_Header_Footer.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO16].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextViewCursorSupplier interface: Cf. [IDLRef41].

XNameAccess interface: Cf. [IDLRef46].

XStyleFamiliesSupplier interface: Cf. [IDLRef79].

The changes the macro does:

The macro creates a header line, as seen in figure 39,

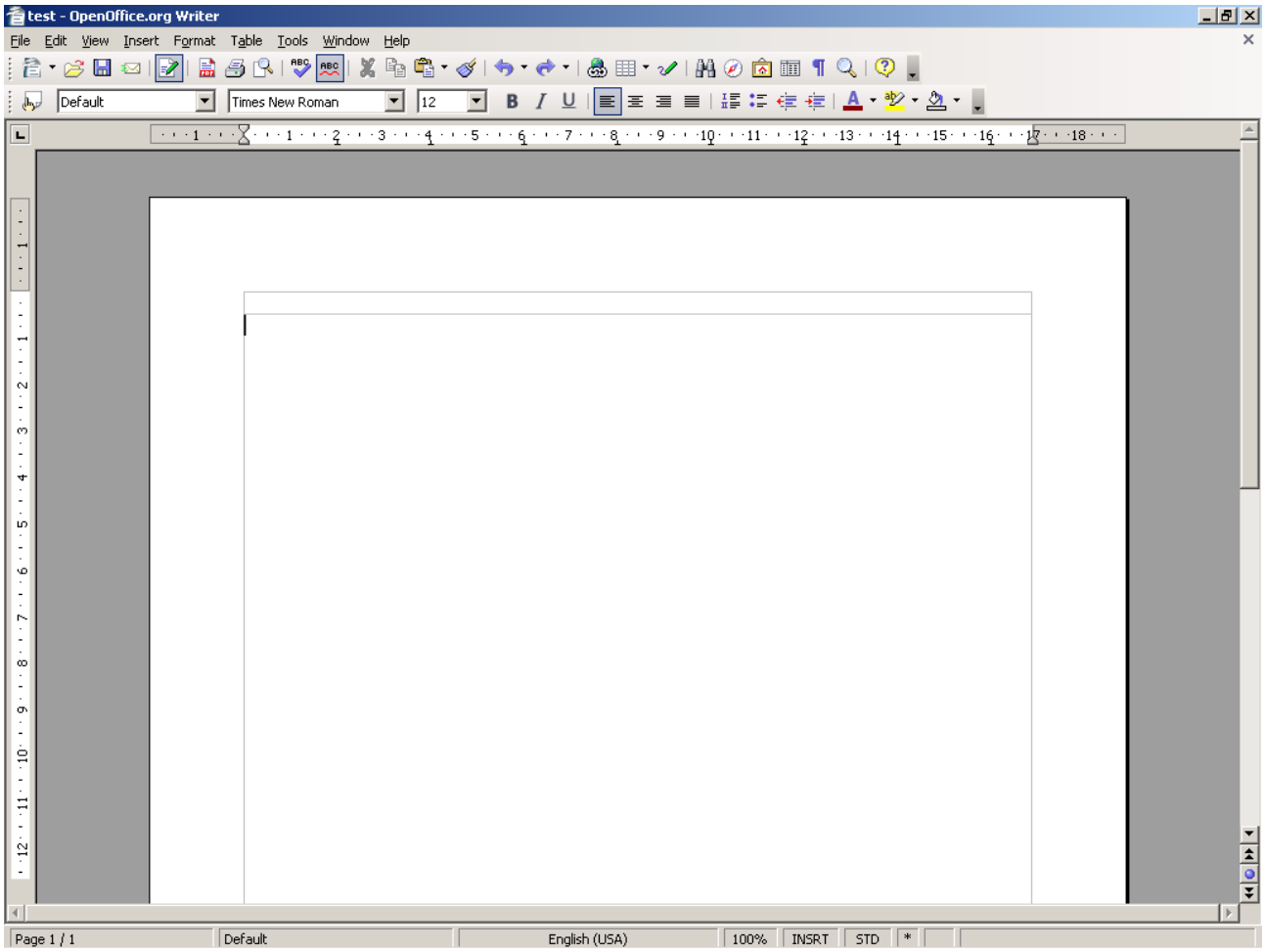

Figure 39: Created header of a writer document.

and a footer line, as depicted in figure 40.

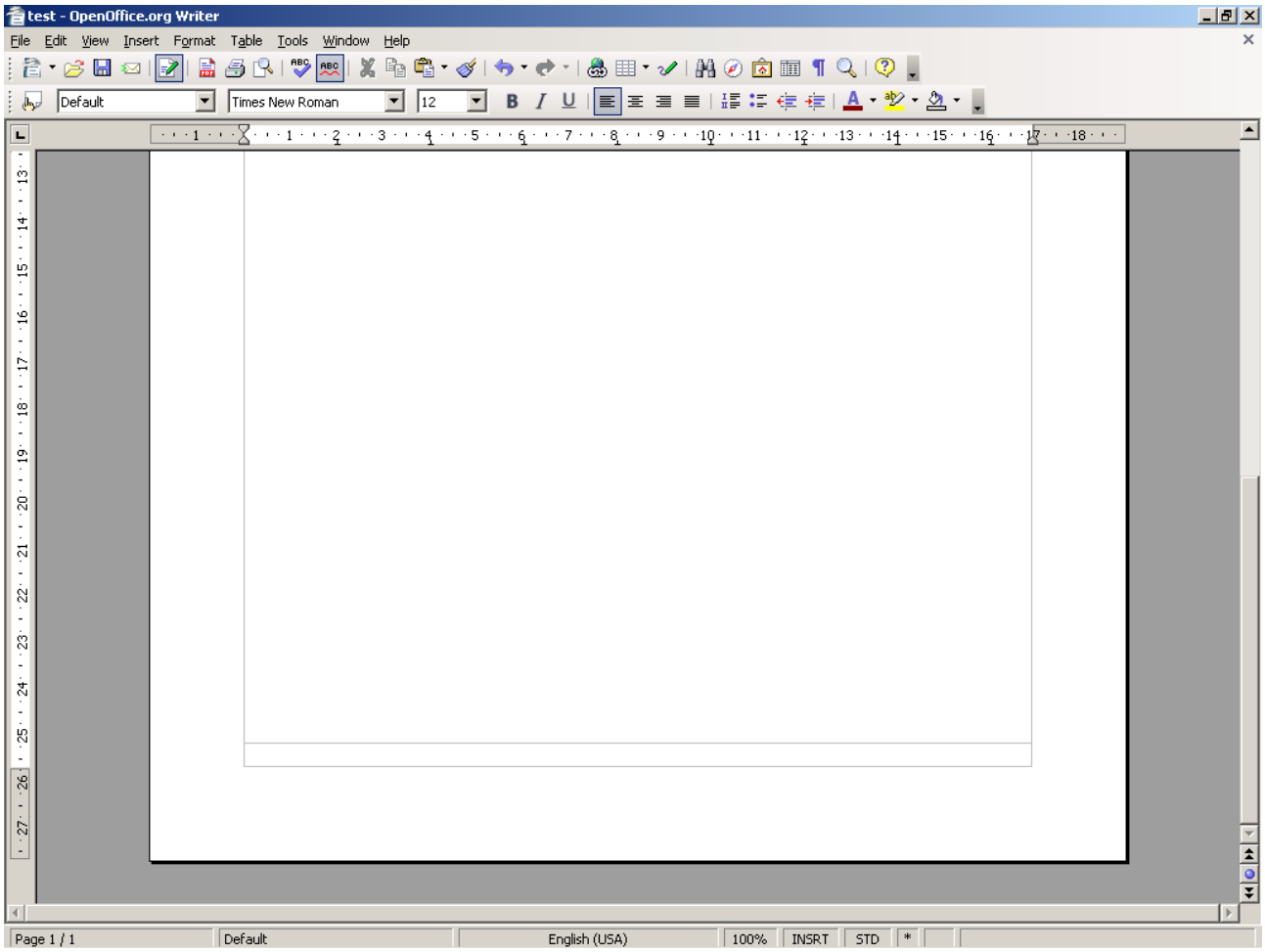

Figure 40: Created footer of a writer document.

6.2.8 Example 8: Insert a Note Field

- Task of the macro: insert a note field at the text cursor position.
- Peculiarities: generating a creation date for the note field.
- Possible solution: creating a new "`Annotation`" object, which represents the note and a "`Date`" structure to describe the date of creation.

This example is inserting a predefined note field to the text at the text cursors current position. First the documents text and current cursor object have to be retrieved again. Next the documents "`XMultiServiceFactory`" interface is used to create a new "`Annotation`" object. In the next step the annotations "`Author`" and "`Content`" properties are filled with string values. The newly created note object also contains a "`Date`" property which needs

a "Date" structure as its value. The current date is retrieved by the ooRexx built in function "DATE" and parsed to a day, month and year value. These values can now be applied to the "Date" structure which in turn can be applied to the annotations "Date" property. Finally the note object is inserted to the documents text using the "insertTextContent" method of the text service. This method requires the current cursor as first, the note objects "XTextContent" interface as second, and a boolean true value as third parameter. Now the note has been added at the current cursor position and the macro ends.

w_Note.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- get the text of the current document
x_TextDocument = x_Document~XTextDocument
x_Text = x_TextDocument~getText

-- get the current cursor position
s_CurrentController = x_TextDocument~getCurrentController()
x_TextViewCursorSupplier = s_CurrentController~XTextViewCursorSupplier
x_CurrentCursor = x_TextViewCursorSupplier~getViewCursor()

-- get current text cursor
x_TextCursor = x_Text~createTextCursorByRange(x_CurrentCursor~getStart())

-- create a factory to create a note object
x_MultiServiceFactory = x_Document~XMultiServiceFactory
s_Annotation = x_MultiServiceFactory~createInstance("com.sun.star.text.TextField.Annotation")

-- get access to the note object properties and fill them
x_AnnotationProperties = s_Annotation~XPropertySet
x_AnnotationProperties~setPropertyValue("Author", "JF")
x_AnnotationProperties~setPropertyValue("Content", "Thats right, Mr. Pitonyak")

-- get current date from rexx and parse, year month and day
currentdate = DATE("S",,,"/")
PARSE VAR currentdate year "/" month "/" day

-- create date object and set its date properties, then add the date object
-- to the note object
date = .bsf~new("com.sun.star.util.Date")
date~bsf.setFieldValue("Day", day)
```

```
date~bsf.setFieldValue("Month", month)
date~bsf.setFieldValue("Year", year)
x_AnnotationProperties~setPropertyValue("Date", date)

-- finally insert the note object into the text
x_TextContent = s_Annotation~XTextContent
x_Text~insertTextContent(x_TextCursor, x_TextContent, .true)

::requires UNO.CLS
```

Sourcecode 25: w_Note.rex

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO17].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Text Document: Cf. [IDLRef36], [IDLRef37], [IDLRef38].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextViewCursorSupplier interface: Cf. [IDLRef41].

XTextContent interface: Cf. [IDLRef89].

Example specific references: Cf. [IDLRef97], [IDLRef98].

setFieldValue(): Cf. [REFBSF05].

The output of this macro:

Figure 41 shows the inserted note field[36] in a Writer document.

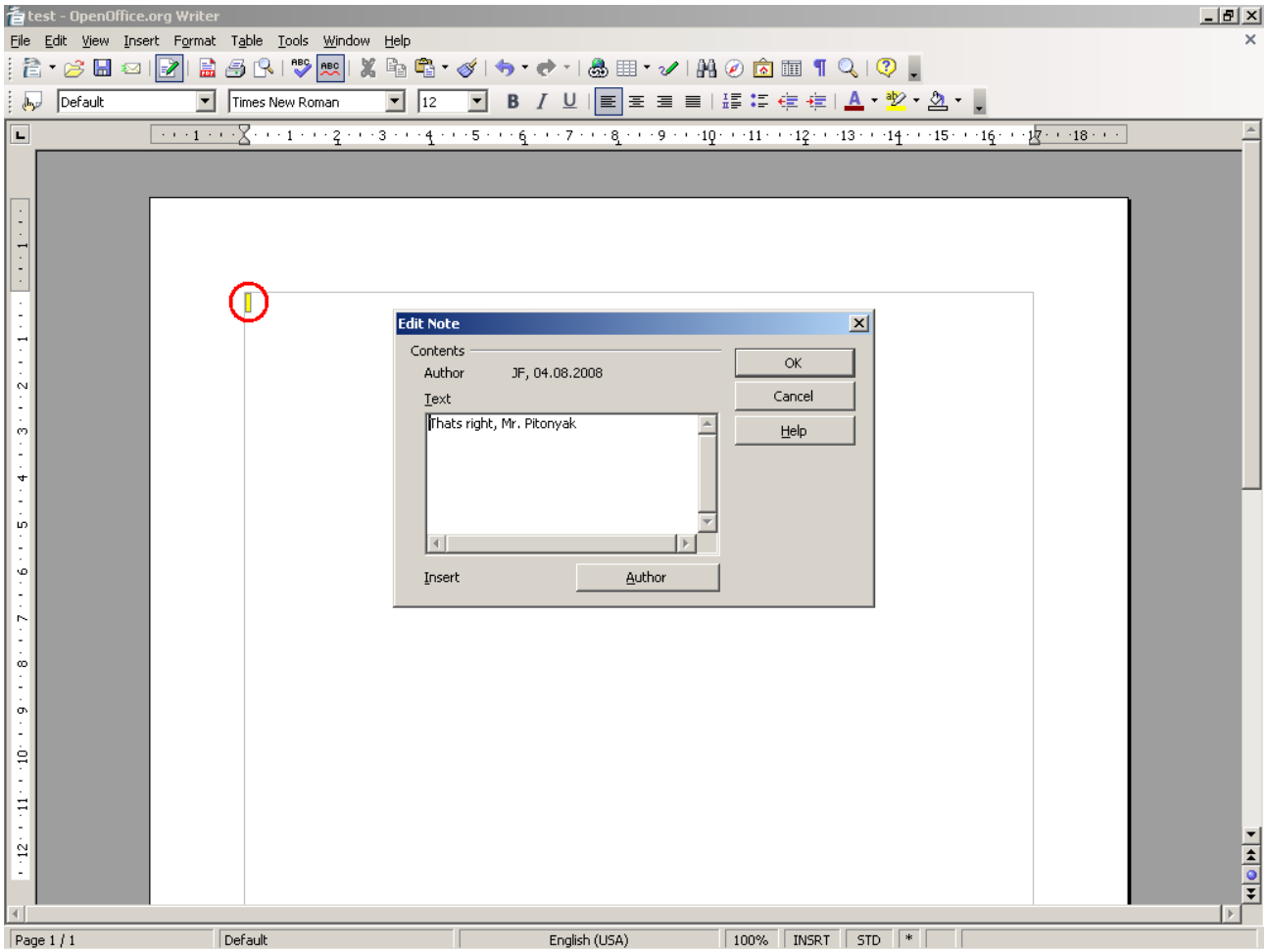

Figure 41: w_Note.rex example.

6.2.9 Example 9: Counting Words using ooRexx

- Task of the macro: count the words of the selected text.
- Peculiarities: usage of the ooRexx method "`WORDS`" to count the selected words.
- Possible solution: get the selected text using the documents "`XModel`" interface and count the words of each selected string.

There are many different examples in Mr. Pitonyaks article how to count the words in a Writer document. This example shows how easy this task is using ooRexx and its built in word counting method "`WORDS`".

[36] The yellow field surrounded by the red circle states the annotation. By double clicking on it, the description window appears.

First a counter variable is created and its value set to zero. Then the current selection of the documents "`XModel`" interface is retrieved. If the returned container is a valid object then some text parts have been selected. Now the "`XIndexAccess`" interface of the container is used to iterate trough the selected text parts and to store them to a string variable by querying the "`XTextRange`" interface of the text parts and call its "`getString`" method. The previously introduces "`WORDS()`" function of ooRexx is now used to count the words of the string and add the result to the counter variable. Finally the counter variable is displayed by a message box.

w_WordCount.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end




-- really easy example, using ooRexx!!!

-- counter variable
wordcount = 0

-- get access to the current selection
x_Model = x_Document~XModel
s_Container = x_Model~getCurrentSelection()


if s_Container <> .nil then
do
   -- if at least one selection has been made:

   -- get an index list of the selection
   x_IndexAccess = s_Container~XIndexAccess
   size = x_IndexAccess~getCount()

   -- iterate trough the selections and retrieve the selected strings
   -- count the words within the strings by using ooRexx builtin function WORDS
   do counter = 1 to size
      s_text = x_IndexAccess~getByIndex(counter - 1)
      x_TextRange = s_text~XTextRange

      wordstring = x_TextRange~getString()

      wordcount = wordcount + WORDS(wordstring)
```

```
   end
end

-- output of counted words
.bsf.dialog~messageBox("Counted Words: " || wordcount, "WordCount", "information")

::requires UNO.CLS
```

Sourcecode 26: w_WordCount.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO18].

XModel interface: Cf. [IDLRef17].

XIndexAccess interface: Cf. [IDLRef92].

XTextRange interface: Cf. [IDLRef96].

Messagebox: Cf. [REFBSF01].

Visual output of this macro:

The messagebox, returned by the macro, contains the number of counted words, which is depicted in figure 42.

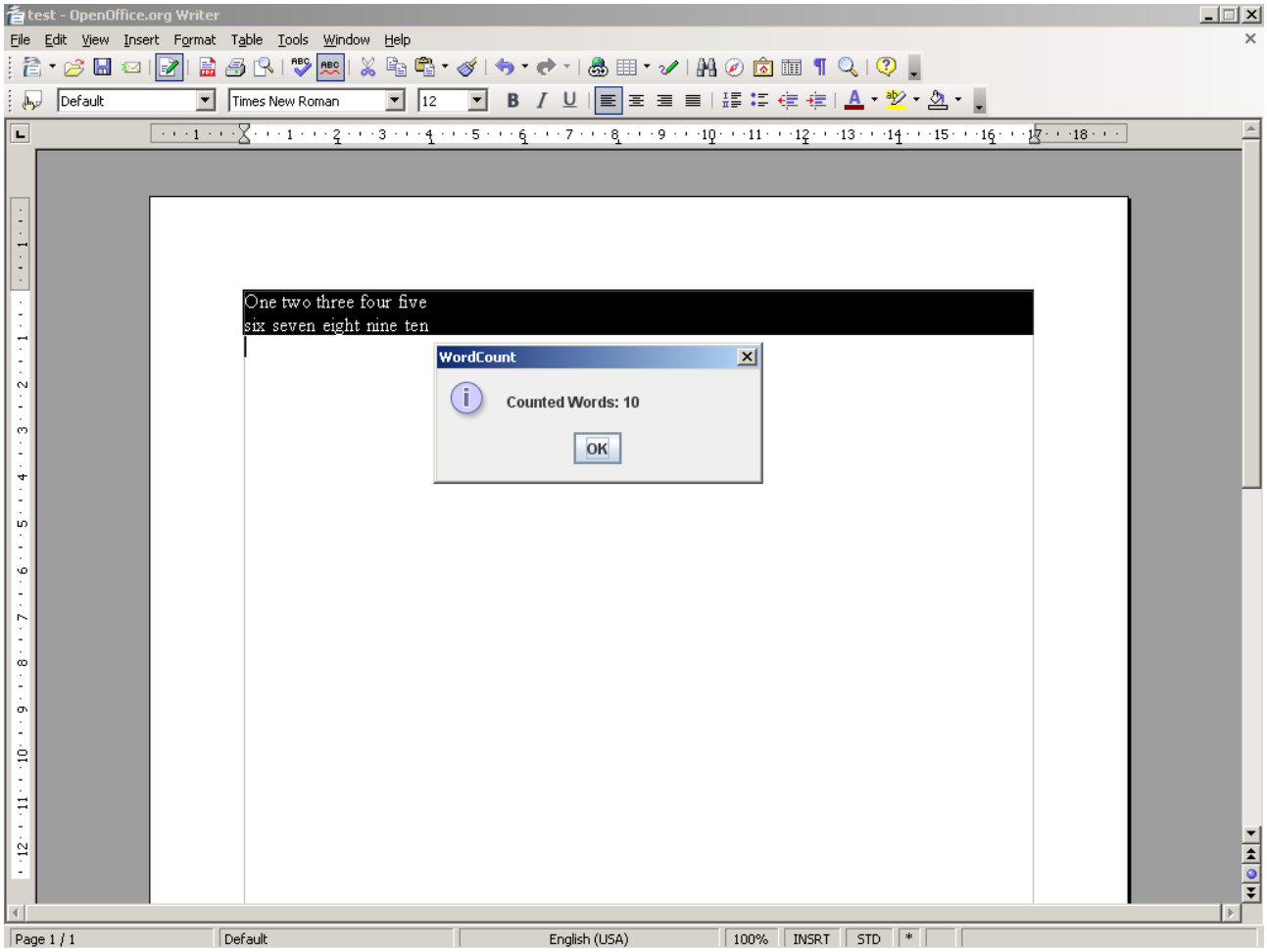

Figure 42: w_WordCount.rex example.

6.3 Calc examples

The following examples of this chapter are examples that use the Calc Program of open office to make changes to the sheets, cells, and graphical objects of a spreadsheet document.

6.3.1 Example 1: Toggle Automatic Calculation

- Task of the macro: toggle the automatic calculation of a Calc document on and off.
- Peculiarities: get access to the automatic calculation setting.

- Possible solution: usage of the documents "`XCalculatable`" interface.

This is a really easy example. It shows how to toggle the automatic calculation on and off. This is useful if a lot of cells must be changed and every change would lead to a recalculation. If the document is a spreadsheet document it also inherits an "`XCalculatable`" interface. The "`isAutomaticCalculationEnabled`" function of this interface is used to determine whether the automatic calculation is on or off. If it is turned off the script turns it on by using the interfaces "`enableAutomaticCalculation`" method providing a boolean value of true as its only parameter. To turn it off again the same method is called providing a false boolean value as parameter.

c_AutoCalc.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- generate interface to automatic calculation entry
x_Calculatable = x_Document~XCalculatable

-- if Calculation is enabled - disable it, else enable it
if x_Calculatable~isAutomaticCalculationEnabled()
then x_Calculatable~enableAutomaticCalculation(.false)
else x_Calculatable~enableAutomaticCalculation(.true)

::requires UNO.CLS
```

Sourcecode 27: c_AutoCalc.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO19].

Example specific references: Cf. [IDLRef99].

Screenshots of the macro:

Figure 43 shows a screenshot of the AutoCalculate setting, before executing the macro,

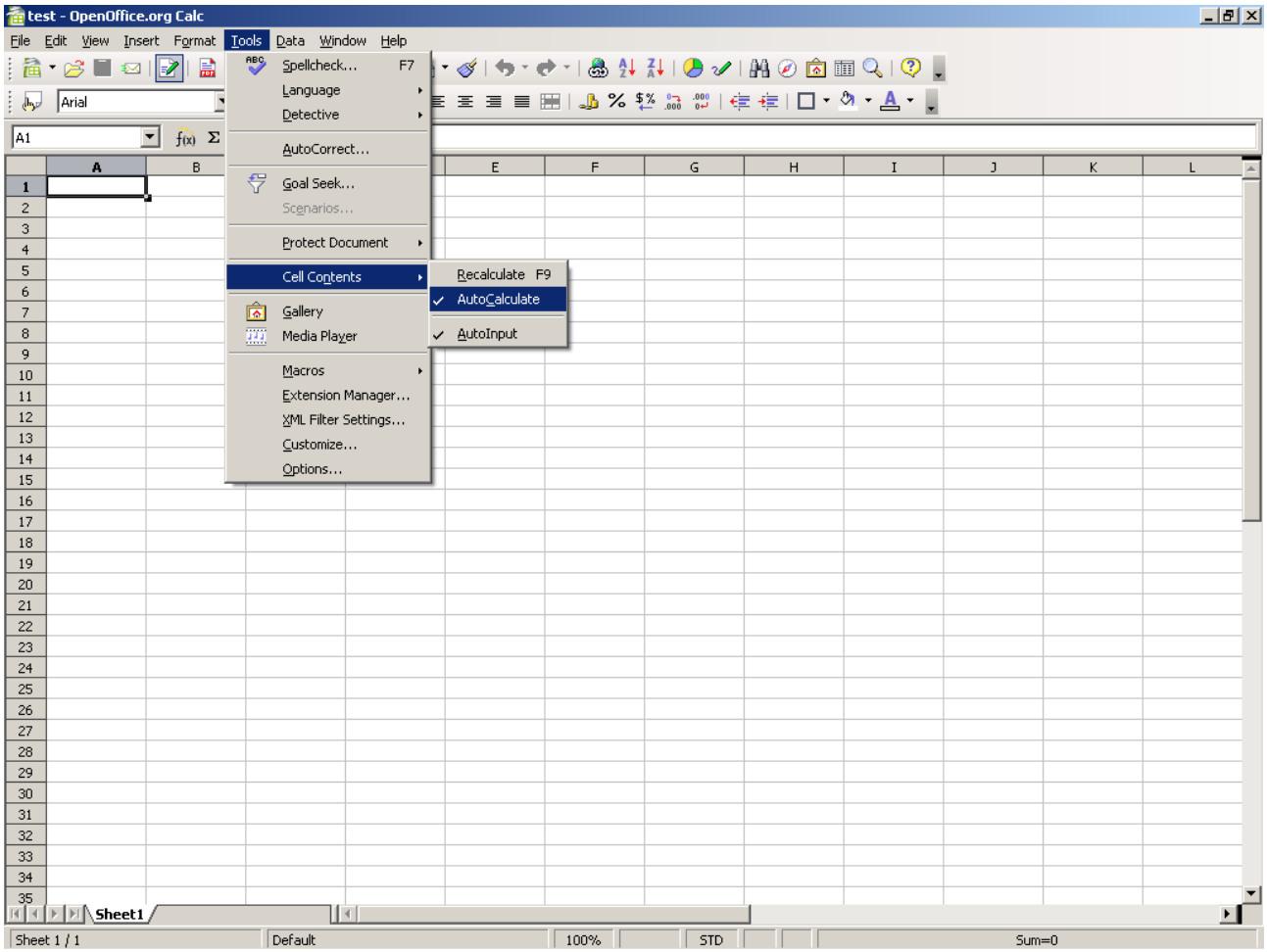

Figure 43: Automatic calculation enabled.

and figure 44 shows it after execution.

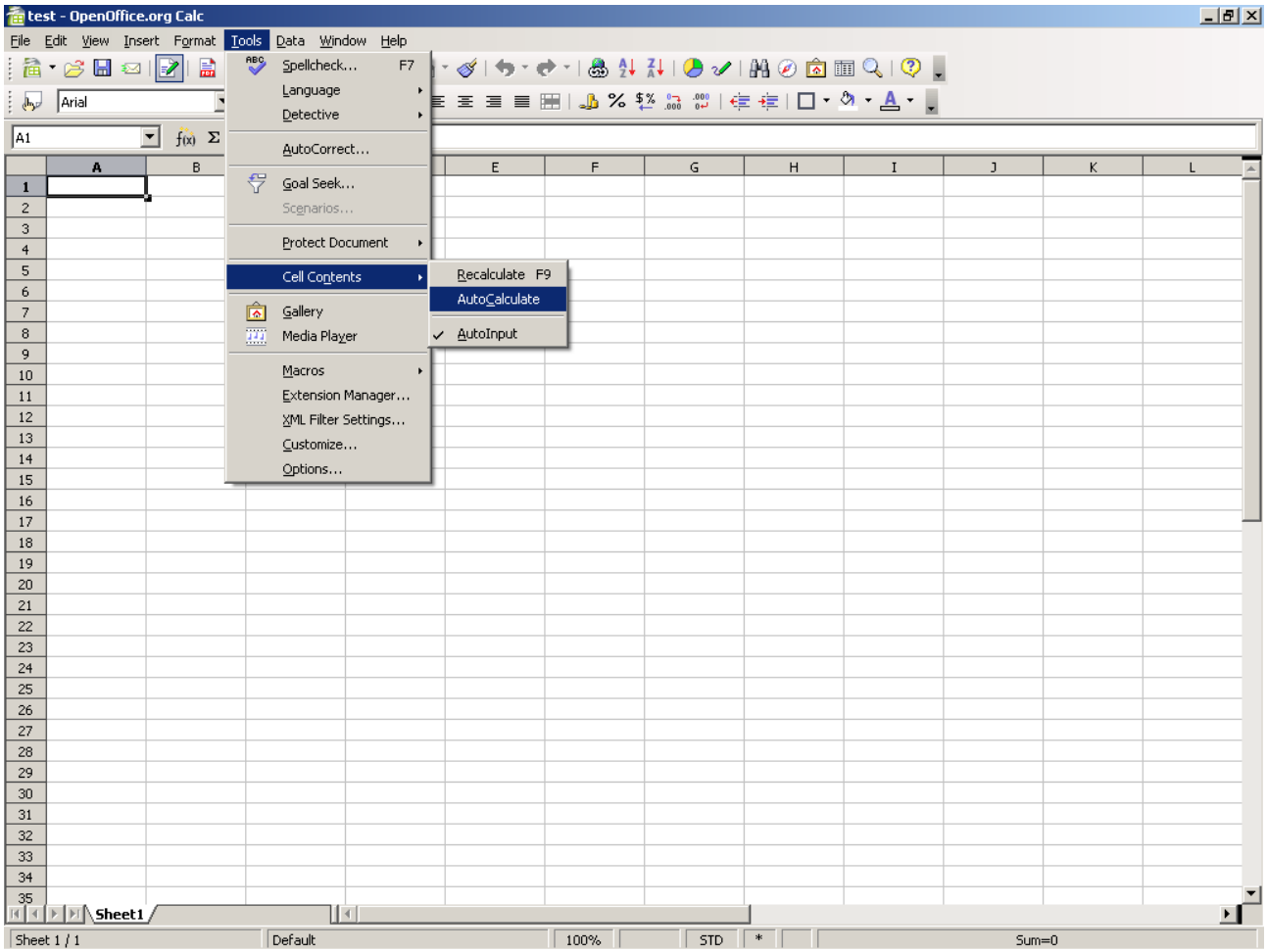

Figure 44: Automatic calculation disabled.

6.3.2 Example 2: Clear the Selected Cells

- Task of the macro: completely delete the selected cells.
- Peculiarities: get the selection and delete the cells.
- Possible solution: calling the "`getCurrentSelection`" method of the documents "`XModel`" interface to get the selected cells, and the "`clearContents`" method of the selections "`XSheetOperation`" interface to clear the cells.

To clear a selection of cells of an Open Office spread sheet, the documents current selection must be acquired by calling the "`getCurrentSelection`" method of the documents "`XModel`" interface. Then the "`clearContents`" method of the selections "`XSheetOperation`" interface is called. This method requires a number as its parameter,

which states what shall be deleted. The number is just a container of bit flags. In this example we want to clear all attributes of the selected cells and therefore we specify all flags.

c_ClearSelection.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- frist get the current selection
x_Model = x_Document~XModel
selection = x_Model~getCurrentSelection()

-- then clear all cell flags (= deleting all content)
x_SheetOp = selection~XSheetOperation
x_SheetOp~clearContents(1+2+4+8+16+32+64+128+256+512)

::requires UNO.CLS
```

Sourcecode 28: c_ClearSelection.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO20].

XModel interface: Cf. [IDLRef17].

Example specific references: Cf. [IDLRef100].

Visual output of this macro:

To delete data, first a spreadsheet with some sample data in it[37] is needed. Such a spreadsheet is depicted in figure 45.

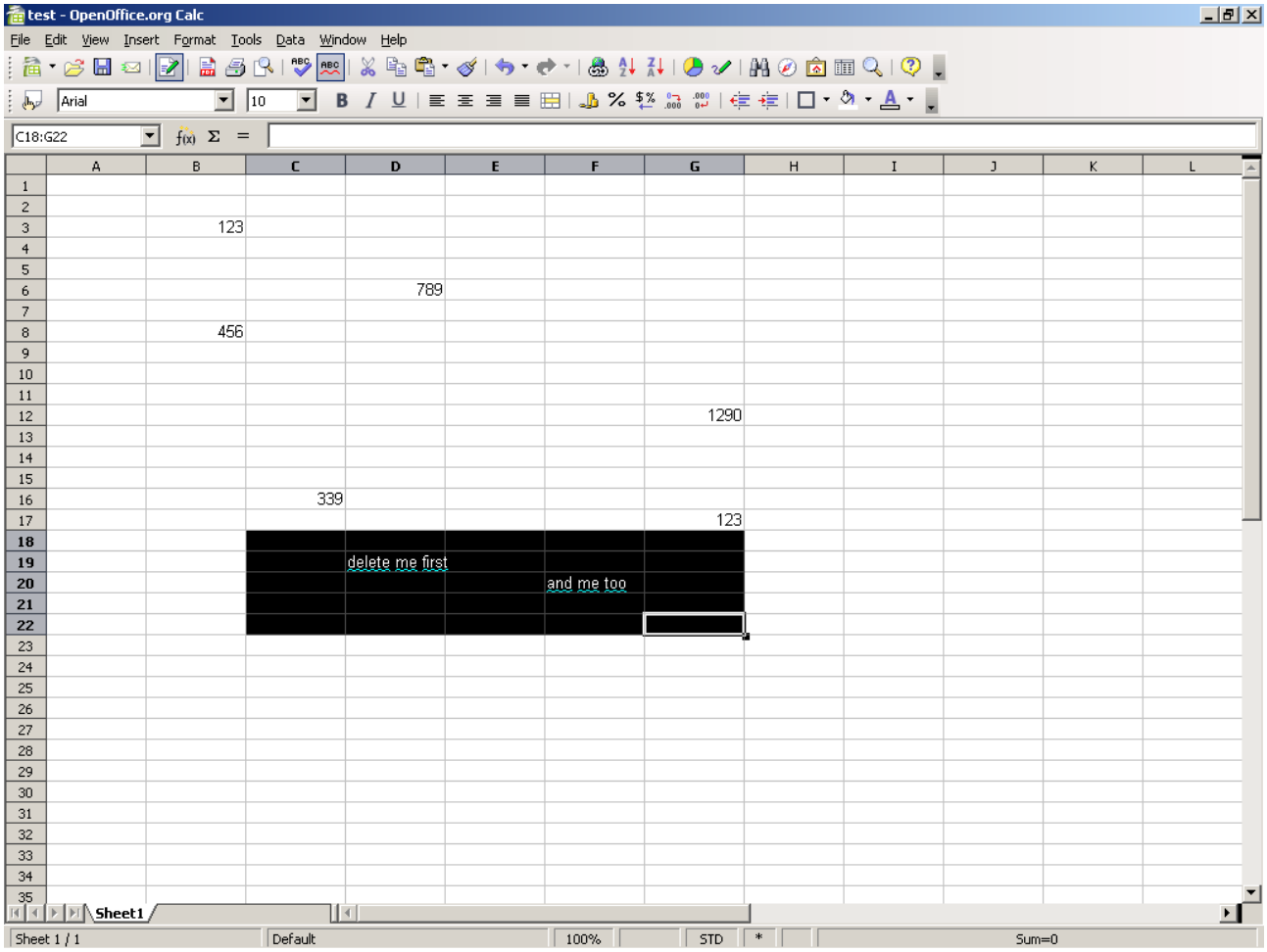

Figure 45: Spreadsheet with test data.

[37] This sheet was also used in the next example.

Figure 46 shows the sample spreadsheet after the selected cells have been deleted by the macro.

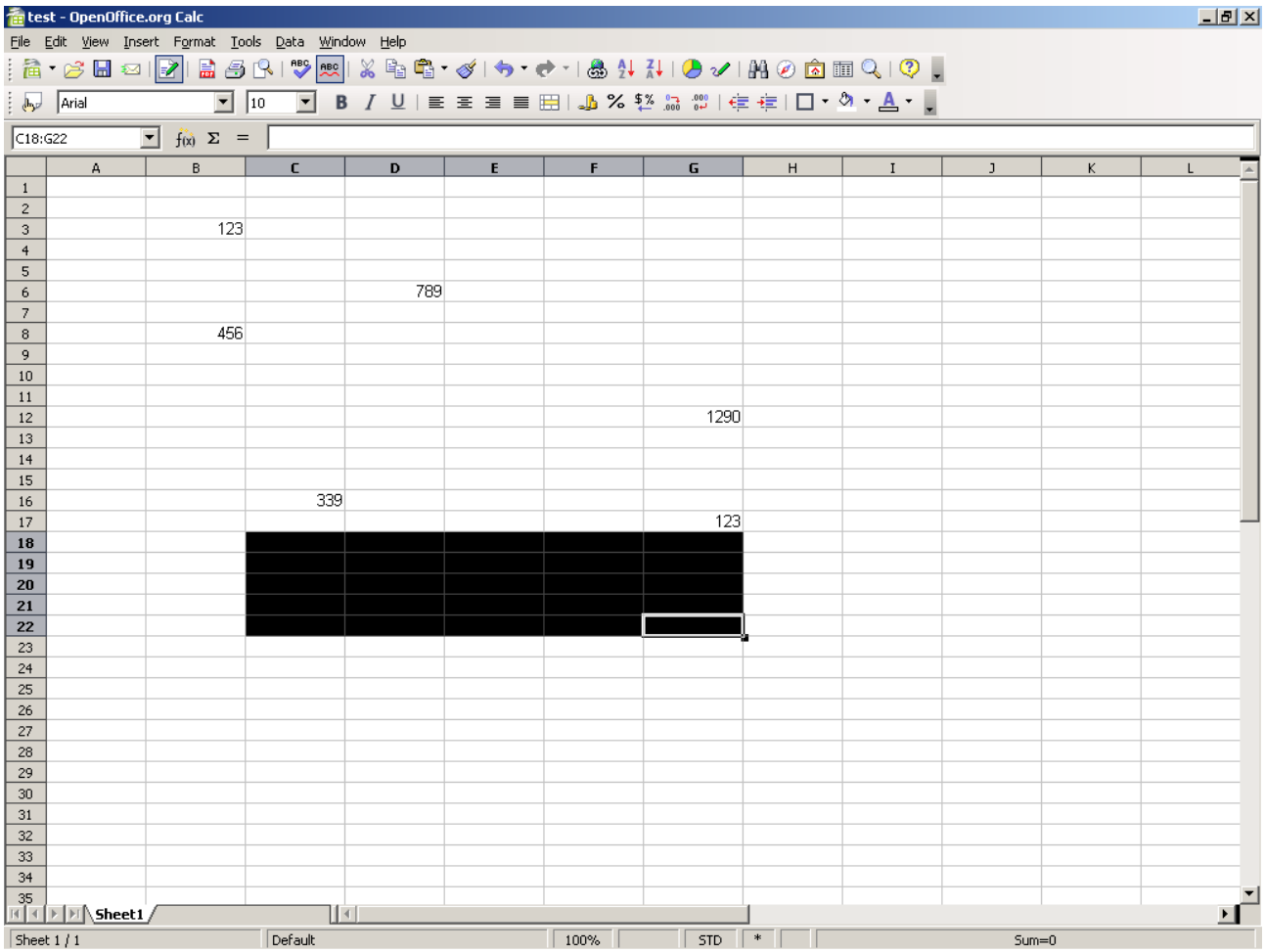

Figure 46: Deleted selection.

6.3.3 Example 3: Clear the Whole Sheet

- Task of the macro: completely delete the active sheet.
- Peculiarities: selecting all cells containing information.
- Possible solution: usage of the "`XUsedAreaCursor`" on a selection service to select all cells containing information and deleting the selected cells content by using the "`XSheetOperation`" interface.

This example is quite similar to the previous example, but now we want to delete the whole sheet. To do this first the currently active spreadsheet has to be determined. This is done by getting the current controller of the documents "`XModel`" interface and retrieving

the "XSpreadsheetView" interface of the current controller to call its "getActiveSheet" method. Now a virtual cursor object of this sheet is created and its "XUsedAreaCursor" interface used to select the area that is containing all the filled cells of this sheet. The parameter of the "gotoStartOfUsedArea" and "gotoEndOfUsedArea" method states whether the selected area should be expanded or not. Finally the selected cells are deleted by using the "XSheetOperation" interface of the virtual cursor the same way as described in the example above.

c_ClearPage.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first get the currently selected sheet
x_Model = x_Document~XModel
s_CurrentController = x_Model~getCurrentController()
x_View = s_CurrentController~XSpreadsheetView
x_Spreadsheet = x_View~getActiveSheet()

-- now create a virtual cursor on this sheet
-- with UsedAreaCursor Interface we just select the region with filled cells
x_SheetCellCursor = x_Spreadsheet~createCursor()
x_UsedAreaCursor = x_SheetCellCursor~XUsedAreaCursor
x_UsedAreaCursor~gotoStartOfUsedArea(.false)
x_UsedAreaCursor~gotoEndOfUsedArea(.true)

-- finally delete all cells and their properties in region
x_SheetOp = x_SheetCellCursor~XSheetOperation
x_SheetOp~clearContents(1+2+4+8+16+32+64+128+256+512)

::requires UNO.CLS
```

Sourcecode 29: c_ClearPage.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO20].

XModel interface: Cf. [IDLRef17].

Controller: Cf. [IDLRef39], [IDLRef40].

Example specific references: Cf. [IDLRef100], [IDLRef101], [IDLRef102], [IDLRef103].

Visual output of this macro:

Using this macro on the example data sheet from above, the macro cleared the whole sheet, as depicted in figure 47.

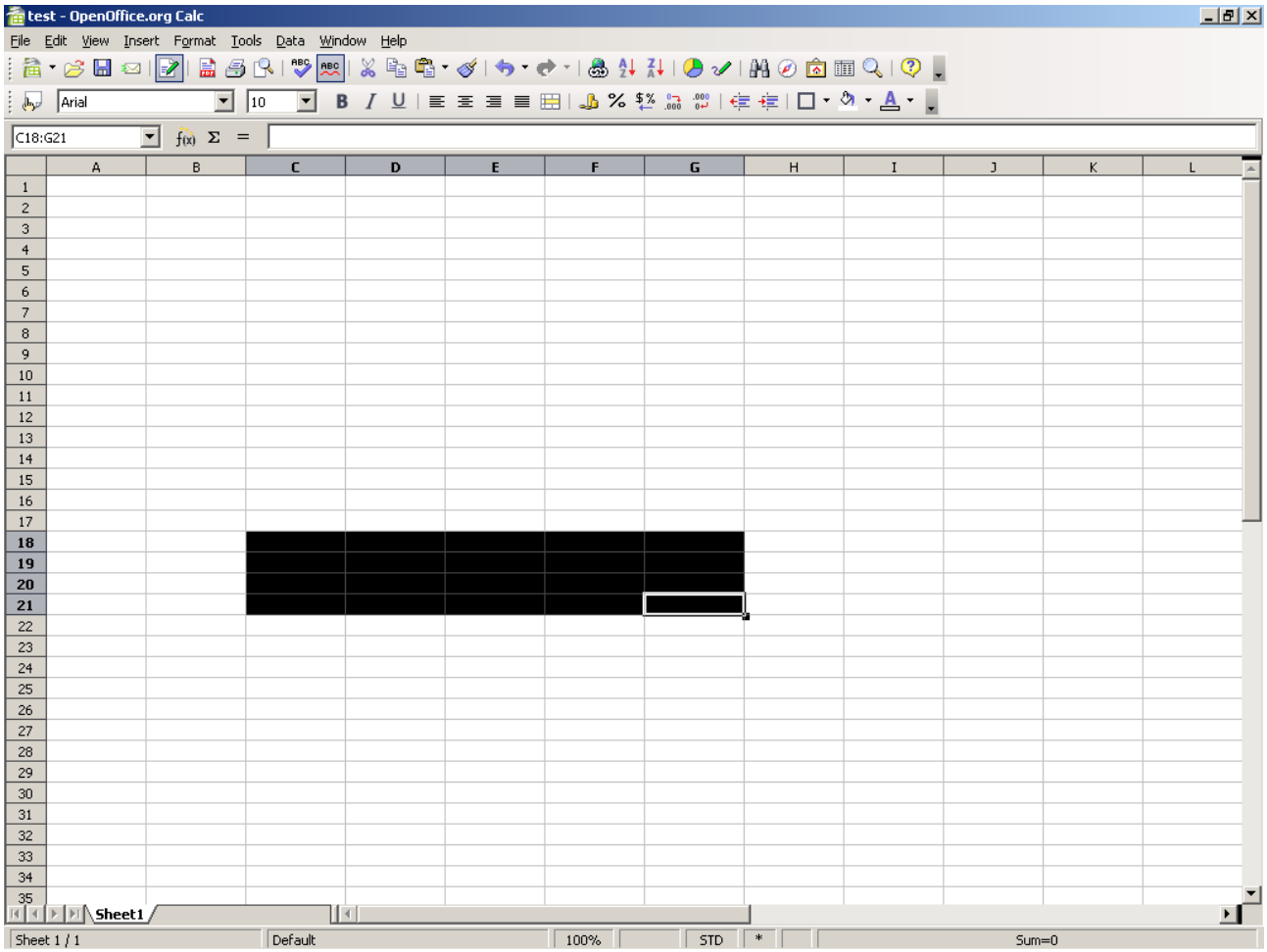

Figure 47: Deletion of the whole sheet.

6.3.4 Example 4: Draw a Shape

- Task of the macro: draw a red line across the sheet.
- Peculiarities: creating a shape.
- Possible solution: create a "`LineShape`" object and draw it on the sheet using the spreadsheets "`DrawPage`" service.

This example draws a big red line on the currently active sheet. Again the current controller of the documents "`XModel`" interface retrieved and its "`XSpreadsheetView`" interface used to receive the active spreadsheet. Then the documents "`XMultiServiceFactory`" is used to create a new shape object of type "`LineShape`". To set the shapes color and width its properties "`LineColor`" and "`LineWidth`" are modified. To set the shapes position and size a "`Size`" and a "`Point`" structure is needed. By calling the "`setSize`" and "`setPosition`" methods, which are provided by the "`XShape`" interface of the shape object, the previously defined Size and Point structures are adopted to the shape. Finally the shape object must be added to the spreadsheets "`DrawPage`" service, which is accessed by calling the "`getDrawPage`" method of the spreadsheets "`XDrawPageSupplier`" interface. Now the "`add`" method of the "`DrawPage`" service is called and the "`XShape`" interface is submitted as its only parameter.

c_DrawLineShape.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end




-- first get currently selected sheet
x_Model = x_Document~XModel
```

```
s_CurrentController = x_Model~getCurrentController()
x_View = s_CurrentController~XSpreadsheetView
x_Spreadsheet = x_View~getActiveSheet()

-- then get the drawpage (used for paintings) of the sheet
x_DrawPageSupplier = x_Spreadsheet~XDrawPageSupplier
x_DrawPage = x_DrawPageSupplier~getDrawPage()

-- create a new Shape and configure it
x_MultiServiceFactory = x_Document~XMultiServiceFactory
s_Shape = x_MultiServiceFactory~createInstance("com.sun.star.drawing.LineShape")

shapecolor = .bsf~new("java.lang.Integer", X2D("FF0000"))
shapeProperties = s_Shape~XPropertySet
shapeProperties~setPropertyValue("LineColor", shapecolor)
linelength = .bsf~new("java.lang.Integer", 500)
shapeProperties~setPropertyValue("LineWidth", linelength)

size = .bsf~new("com.sun.star.awt.Size")
size~bsf.setFieldValue("width", 25000)
size~bsf.setFieldValue("height", 14000)

pos = .bsf~new("com.sun.star.awt.Point")
pos~bsf.setFieldValue("X", 400)
pos~bsf.setFieldValue("Y", 400)

x_Shape = s_Shape~XShape
x_Shape~setSize(size)
x_Shape~setPosition(pos)

-- finally add the shape to the sheet
x_DrawPage~add(x_Shape)

::requires UNO.CLS
```

Sourcecode 30: c_DrawLineShape.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO21].

XModel interface: Cf. [IDLRef17].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Controller: Cf. [IDLRef39], [IDLRef40].

Example specific references: Cf. [IDLRef101], [IDLRef103], [IDLRef104], [IDLRef105], [IDLRef106], [IDLRef107], [IDLRef108], [IDLRef109].

setFieldValue(): Cf. [REFBSF05].

The graphical result:

After execution, the macro produced a big red line, as displayed in figure 48 below.

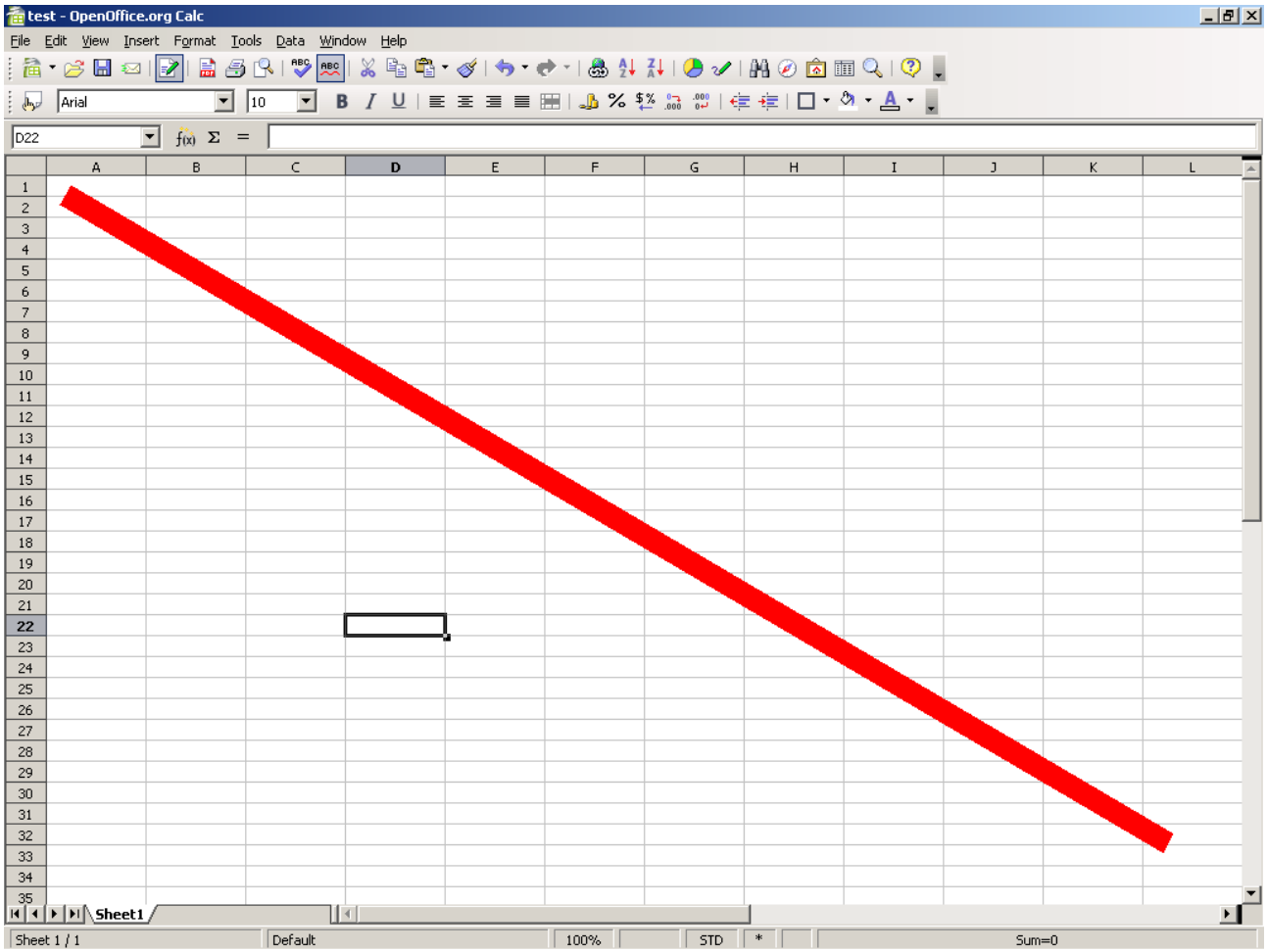

Figure 48: Red line shape.

6.3.5 Example 5: Import a Graphic File

- Task of the macro: import a graphic file into a spreadsheet.
- Peculiarities: creating a graphics object which is able to hold a graphic and can be added to the spreadsheet.
- Possible solution: create a "`GraphicObjectShape`" object, load a graphics file into it and draw it on the sheet using the spreadsheets "`DrawPage`" service.

This example is importing a picture by linking the graphics object to an external file. Even if it is not clear at the first sight, this example runs trough nearly the same procedures as the previous example, because every imported picture is held and displayed by a shape

object.

At the beginning of this example a "`FilePicker`" dialog is used to let the user select a file containing a picture. For more details on the "`FilePicker`" dialog read the example 6.2.2.

First the current spreadsheet, the documents "`XMultiServiceFactory`" interface, as well as the spreadsheets "`DrawPage`" service are needed to be retrieved as described in the previous example. The Factory is then used to create a "`GraphicObjectShape`" service which properties are accessed by querying the services "`XPropertySet`" interface. The "`GraphicURL`" property is set to the URL of the selected file to link the graphics object to the external file. Then a "`Size`" and a "`Point`" structure is created to apply the pictures position and size like in the example above. If the picture is not resized, it is displayed so small, that it can barely be seen. In this example the picture does not keep its original size, instead the picture is resized to a custom height and width. Finally the "`DrawPage`" service is used to add the new picture as already described in the example before.

c_ImportGraphic.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- first we need a file dialog to select a file to import
x_MultiServiceFactory = x_ComponentContext~getServiceManager()~XMultiServiceFactory
s_FileDialog = x_MultiServiceFactory~createInstance("com.sun.star.ui.dialogs.OfficeFilePicker")
x_FileDialog = s_FileDialog~XFilePicker
x_FileDialogFilters = s_FileDialog~XFilterManager

-- adding some file extensions
x_FileDialogFilters~appendFilter("All *.*", "*.*")

-- Better name for our dialog:
x_FileDialog~setTitle("Select Graphic File")

-- selecting more than one file at once dissalowed (to allow it use 1)
x_FileDialog~setMultiSelectionMode(0)
```

```
filechoosen = x_FileDialog~execute()
if ( filechoosen ) then
do
   -- if a file has been choosen:
   -- read filename
   files = x_FileDialog~getFiles()
   file = files[1]

   -- now get the currently slected sheet
   x_Model = x_Document~XModel
   s_CurrentController = x_Model~getCurrentController()
   x_View = s_CurrentController~XSpreadsheetView
   x_Spreadsheet = x_View~getActiveSheet()

   -- and the drawing interface from this sheet
   x_DrawPageSupplier = x_Spreadsheet~XDrawPageSupplier
   x_DrawPage = x_DrawPageSupplier~getDrawPage()

   -- next create a Graphics object (a Shape)
   x_MultiServiceFactory = x_Document~XMultiServiceFactory
   s_Shape = x_MultiServiceFactory~createInstance("com.sun.star.drawing.GraphicObjectShape")
   shapeProperties = s_Shape~XPropertySet

   -- and enter the filename to the link --> JUST A LINKED GRAPHIC!
   shapeProperties~setPropertyValue("GraphicURL", file)

   -- enlarge and reposition the graphical object
   size = .bsf~new("com.sun.star.awt.Size")
   size~bsf.setFieldValue("width", 5535)
   size~bsf.setFieldValue("height", 5535)

   pos = .bsf~new("com.sun.star.awt.Point")
   pos~bsf.setFieldValue("X", 400)
   pos~bsf.setFieldValue("Y", 400)

   x_Shape = s_Shape~XShape
   x_Shape~setSize(size)
   x_Shape~setPosition(pos)

   -- finally add the graphics object to the sheet
   x_DrawPage~add(x_Shape)
end


::requires UNO.CLS
```

Sourcecode 31: c_ImportGraphic.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO22].

XModel interface: Cf. [IDLRef17].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Controller: Cf. [IDLRef39], [IDLRef40].

File Picker dialog: Cf. [IDLRef82], [IDLRef83], [IDLRef84], [UMIFOO25].

Example specific references: Cf. [IDLRef101], [IDLRef103], [IDLRef104], [IDLRef105], [IDLRef107], [IDLRef108], [IDLRef109], [IDLRef110].

setFieldValue(): Cf. [REFBSF05].

The picture insertion:

To insert a picture, first a graphics file needs to be selected. Figure 49 shows the file dialog which enables the user to select a graphic file to insert.

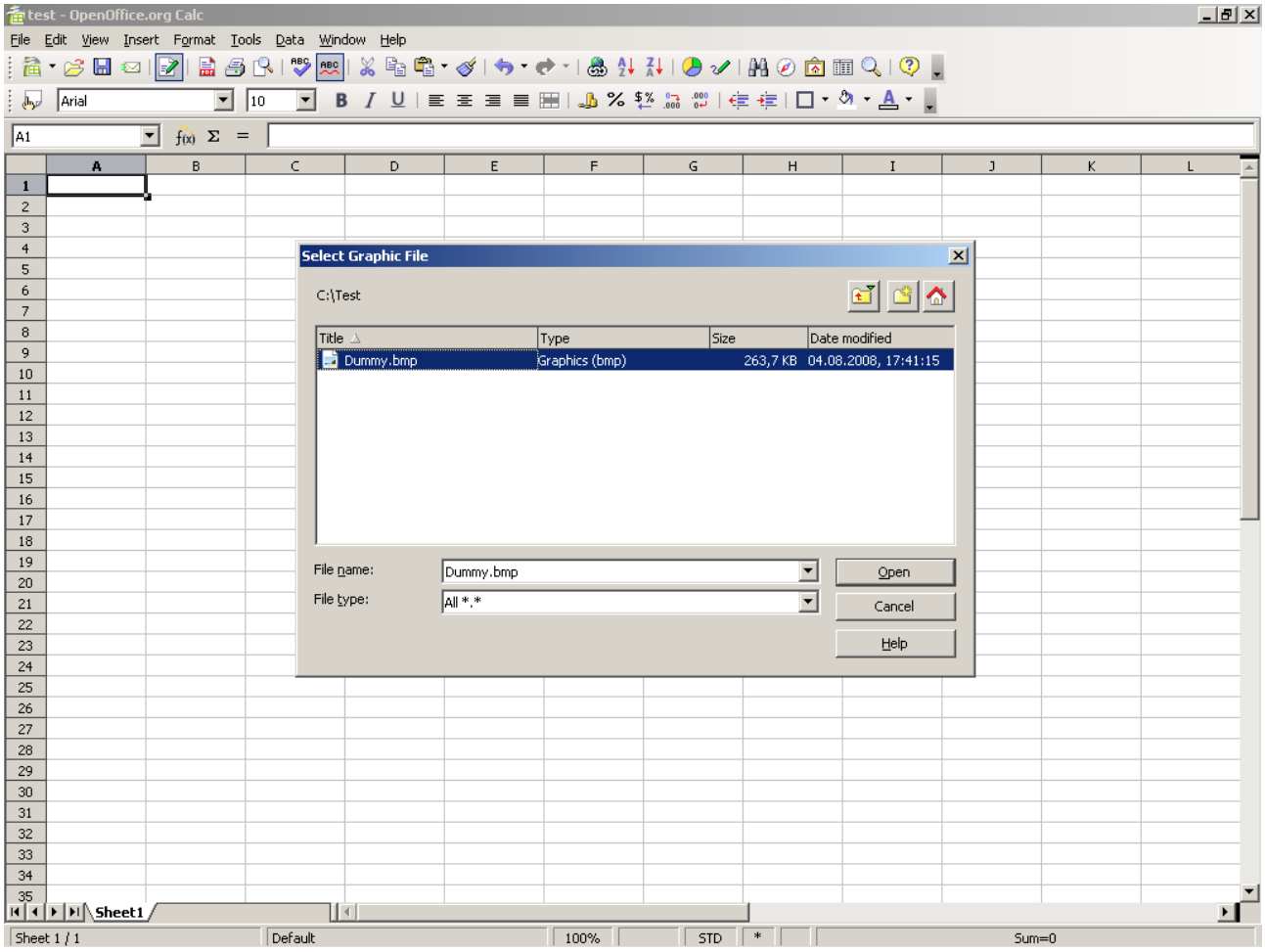

Figure 49: File selection dialog to select graphic file for import.

Figure 50 shows the spreadsheet and the inserted picture in it.

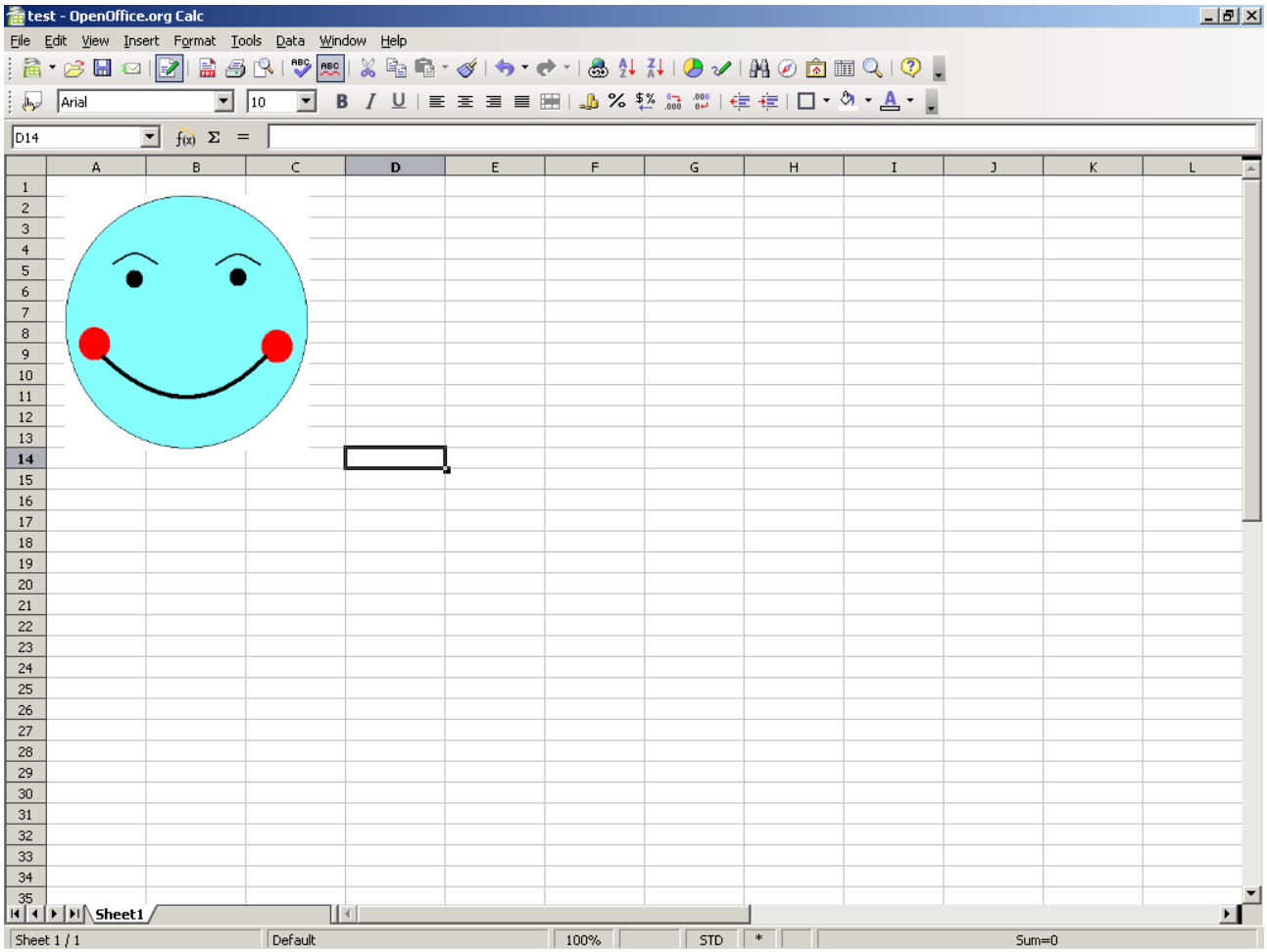

Figure 50: Imported graphic file.

6.3.6 Example 6: Change Text Cell to URL

- Task of the macro: create an "`URL`" text field object out of the cells text content.
- Peculiarities: get a single selected cell.
- Possible solution: To get a single cell out of many selected cells, store the selection to a variable. Then remove the selection and get the cell at the cursor position. Finally restore the selection stored in the variable. To create an URL content create an "`URL`" text field object and apply it to the cells text content.

This example converts a cell containing text information to a cell with an URL link in it. The link will use the cells text as its representation text as well as its target. Therefore the

first task is to read the text of the currently selected cell. To read the cell we need the currently selected cell object.

The problem retrieving the currently selected cell object is, that if more than one cell is selected, no valid single cell information is returned. Therefore a workaround is needed to temporarily remove the selection. This is done by the following steps. First the "`CurrentController`" service from the documents "`XModel`" interface is used to store the current selection to a variable. Then a new "`SheetCellRange`" service is created using the documents "`XMultiServiceFactory`" interface. The new service is applied as the new selection by calling the "`select`" method of the current controllers "`XSelectionSupplier`" interface. In the next step the selection of the "`XModel`" interface is retrieved again and its "`XCellAddressable`" interface is queried. This interfaces "`getCellAddress`" method is then called to retrieve a "`CellAddress`" structure containing the sheet index, as well as the row and the column numbers of the currently selected cell. At the beginning of the example also an indexed list of all Spreadsheets of this document has been created. This list is used now to get the spreadsheet stated in the "`CellAddress`" structure. Finally the "`uno.getCell`" function of the "`UNO.CLS`" module, which will need the spreadsheet as first parameter and the column and row information as second and third parameter, is called to get the cell object.

Now the cells "`Text`" service is available by using the cell objects "`XTextRange`" interface and its text string is stored to a variable. To add a URL a cell a new URL service must be created by the documents "`XMultiServiceFactory`" interface. Its "`Representation`" and "`URL`" properties are set to the previously stored string variable. Finally the cells text content is deleted and the newly created URL text field is inserted as text content of the cell.

c_InsertURL.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
```

```
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



/*
first we need to get the currently selected cell in the currently
selected sheet. If more than one cell is selected we first have to
reduce the amount of selected cells to only one, otherwise an error
occurs.
*/

-- query a list of spreadsheets (used later)
x_SpreadsheetDocument = x_Document~XSpreadsheetDocument
x_Spreadsheets = x_SpreadsheetDocument~getSheets()
x_SpreadsheetIA = x_Spreadsheets~XIndexAccess

-- now get the current selection
x_Model = x_Document~XModel
currentselection = x_Model~getCurrentSelection()

-- now  create a new selection object with only one cell selected
s_CurrentController = x_Model~getCurrentController()
x_MultiServiceFactory = x_Document~XMultiServiceFactory
newselection = x_MultiServiceFactory~createInstance("com.sun.star.sheet.SheetCellRanges")

-- use the new selection on the current sheet
x_View = s_CurrentController~XSelectionSupplier
x_View~select(newselection)

-- get the currently selected cell and its address
noselectioncell = x_Model~getCurrentSelection()

x_CellAddressable = noselectioncell~XCellAddressable
st_CellAddress = x_CellAddressable~getCellAddress()

-- restore old selection
x_View~select(currentselection)

-- get position of current cell and query cell object
currentcell.sheet = st_CellAddress~bsf.getFieldValue("Sheet")
currentcell.column = st_CellAddress~bsf.getFieldValue("Column")
currentcell.row = st_CellAddress~bsf.getFieldValue("Row")

s_Spreadsheet = x_SpreadsheetIA~getByIndex(currentcell.sheet)
x_Spreadsheet = s_Spreadsheet~XSpreadsheet

cell = uno.getCell(x_Spreadsheet, currentcell.column, currentcell.row)

-- create text interface on cell
x_TextRange = cell~XTextRange
x_Text = x_TextRange~getText()

-- read cell textcontent
urlstring = x_Text~getString()

-- create url field
s_urlfield = x_MultiServiceFactory~createInstance("com.sun.star.text.TextField.URL")
urlproperties = s_urlfield~XPropertySet
urlproperties~setPropertyValue("Representation", urlstring)
urlproperties~setPropertyValue("URL", urlstring)

-- clear cell and write urlfield
x_TextContent = s_urlfield~XTextContent
x_Text~setString("")
x_Text~insertTextContent(x_Text~createTextCursor(), x_TextContent, .false)

::requires UNO.CLS
```

Sourcecode 32: c_InsertURL.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO23].

XModel interface: Cf. [IDLRef17].

Propertyset: Cf. [IDLRef20], [IDLRef21].

Controller: Cf. [IDLRef39], [IDLRef40].

XTextContent interface: Cf. [IDLRef89].

XIndexAccess interface: Cf. [IDLRef92].

XTextRange interface: Cf. [IDLRef96].

Example specific references: Cf. [IDLRef111], [IDLRef112], [IDLRef113], [IDLRef114], [IDLRef115], [IDLRef116], [IDLRef117], [REFOOO06].

Screenshots of the conversion:

The macro coverts a cell, filled with text content, to a cell countaining an URL object. Therefore the macro needs a spreadsheet with cell, containing some text, as displayed in figure 51.

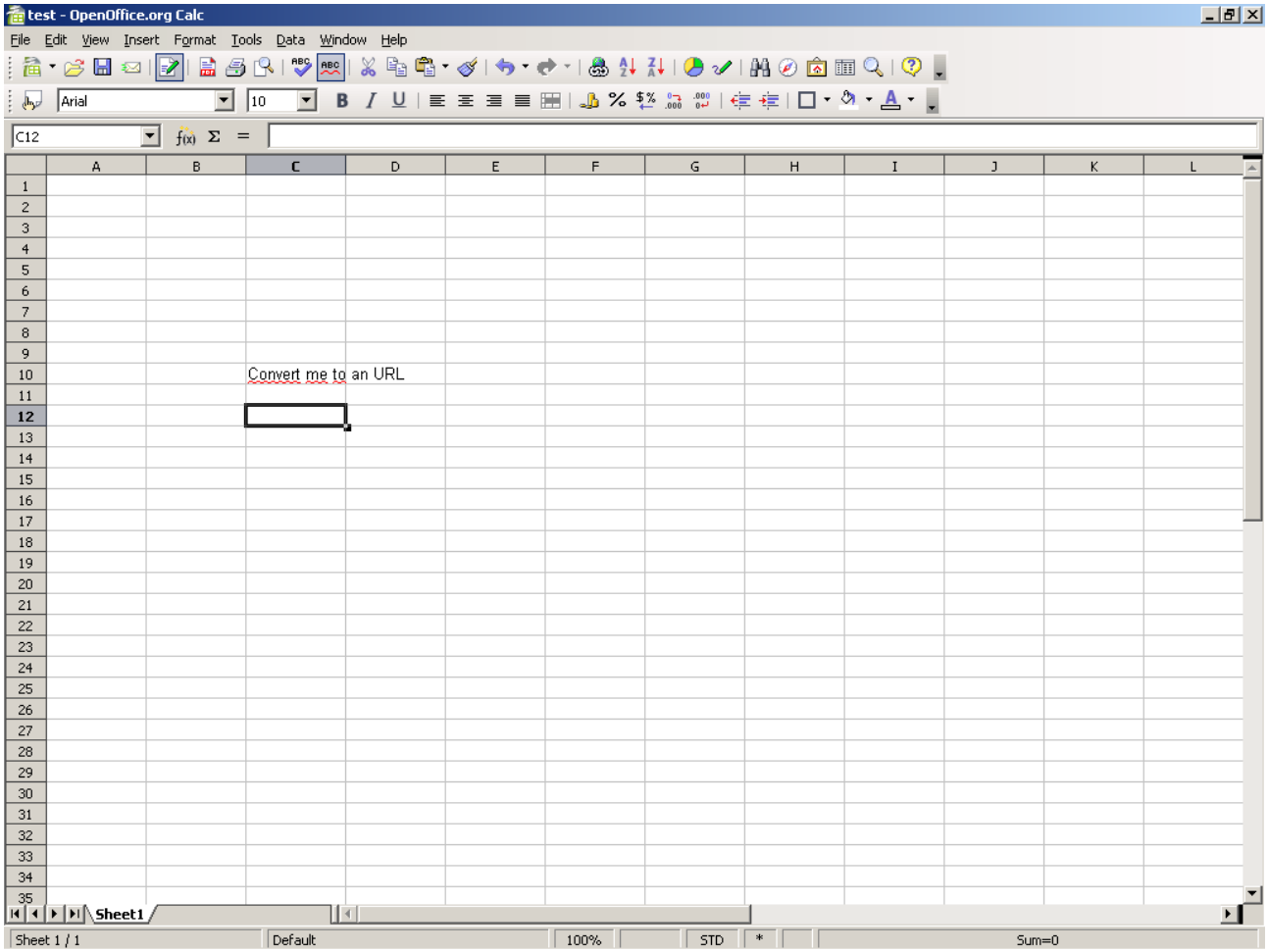

Figure 51: Cell with text content.

After positioning the cell cursor over the text cell and starting the macro, the same cell contains an URL content as shown in figure 52.

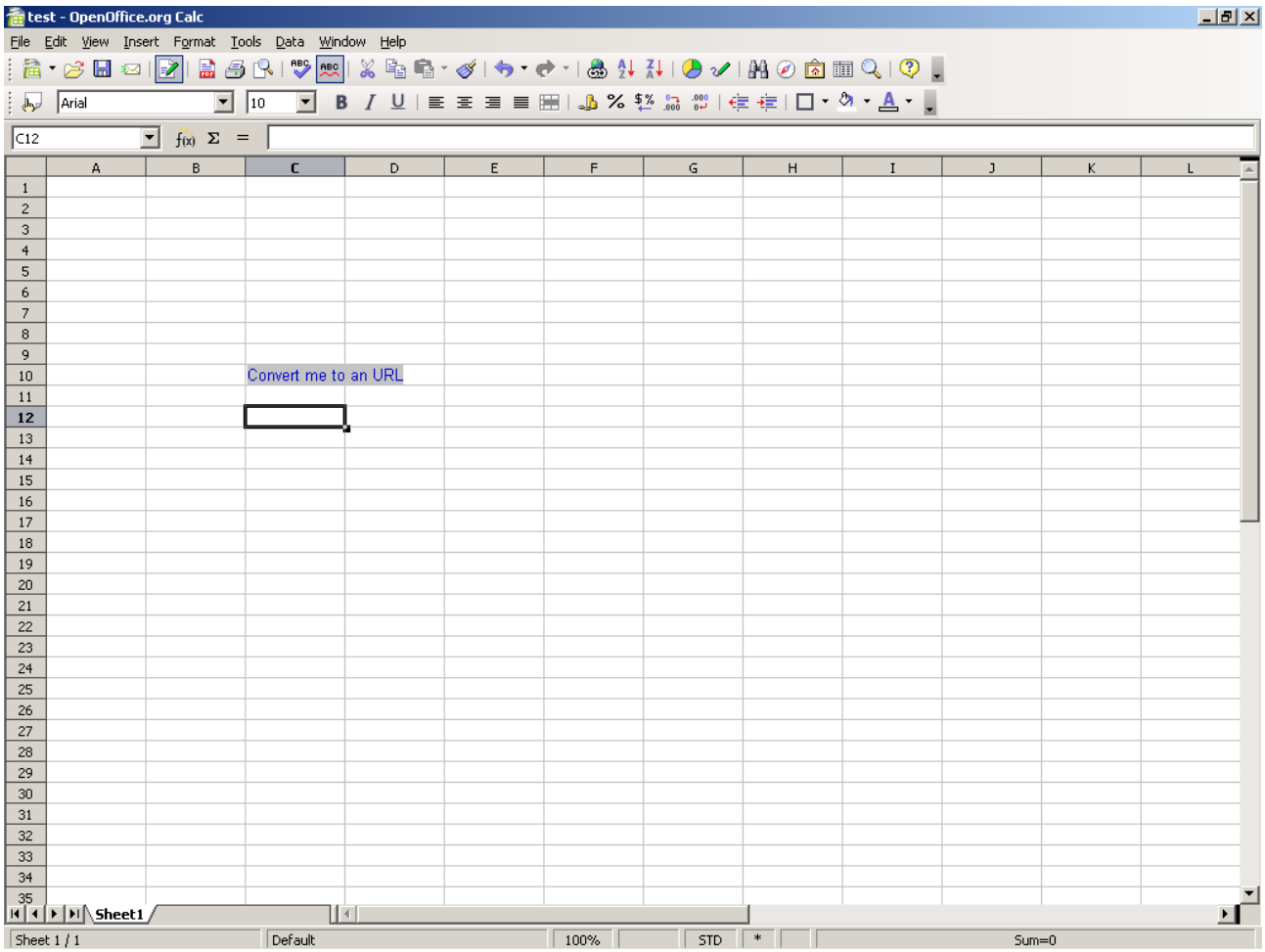

Figure 52: Converted text content (to URL).

6.3.7 Example 7: Merge and Unmerge Cells

- Task of the macro: merge the selected cells, or unmerge them if the selection already contains merged cells.
- Peculiarities: get access to the merge and unmerge functionality of a spreadsheet.
- Possible solution: usage of the selections "`XMergeabel`" interface.

This example shows how to find out if a selection is containing merged cells and merge or unmerge the selected cells. To get access to these functions the "`XMergeabel`" interface of the current selection must be acquired. It can be retrieved by calling the "`getCurrentSelection`" method of the documents "`XModel`" interface. Next the

"`getIsMerged`" function of the "`XMergeabel`" interface is called to find out if the selected area contains one or more merged cells. If there are merged cell, the interfaces "`merge`" method is called adding a boolean false value as parameter to deactivate the merging. If the selection consists of unmerged cells, the cells are merged by calling the "`merge`" method adding a boolean true value as parameter.

c_Merge_Unmerge.rex

```
-- try to get a script context, will be .nil, if script was not invoked by OOo
x_ScriptContext = uno.getScriptContext()
if (x_ScriptContext <> .nil) then
do
   -- invoked by OOo as a macro

   -- get context
   x_ComponentContext = x_ScriptContext~getComponentContext
   -- get desktop (an XDesktop)
   x_Desktop  = x_ScriptContext~getDesktop
   -- get current document
   x_Document = x_ScriptContext~getDocument
end
else
do
   -- called from outside of OOo, create a connection

   -- connect to Open Office and get component context
   x_ComponentContext = UNO.connect()
   -- create a desktop service and its interface
   service = "com.sun.star.frame.Desktop"
   s_Desktop = x_ComponentContext~getServiceManager~XMultiServiceFactory~createInstance(service)
   x_Desktop = s_Desktop~XDesktop
   -- get the last active document
   x_Document = x_Desktop~getCurrentComponent()
end



-- query current selection
x_Model = x_Document~XModel
selection = x_Model~getCurrentSelection()
x_Mergable = selection~XMergeable

-- if slecetion contains a merged cell unmerge it, else merge
-- whole selection to one cell
if x_Mergable~getIsMerged() then
do
   x_Mergable~merge(.false)
end
else
do
   x_Mergable~merge(.true)
end

::requires UNO.CLS
```

Sourcecode 33: c_Merge_Unmerge.rex

References of this macro:

Connection and Script Context related references: Cf. [IDLRef01], [IDLRef02], [IDLRef03], [IDLRef04], [IDLRef05], [IDLRef06], [IDLRef07], [IDLRef08], [REFOOO01], [REFOOO02].

Reference to the example of Mr. Pitonyaks paper: Cf. [UMIFOO24].

XModel interface: Cf. [IDLRef17].

Example specific references: Cf. [IDLRef118].

Visual output of this macro:

Figure 53 shows the spreadsheet with the outcome of the example described above, some merged cells.

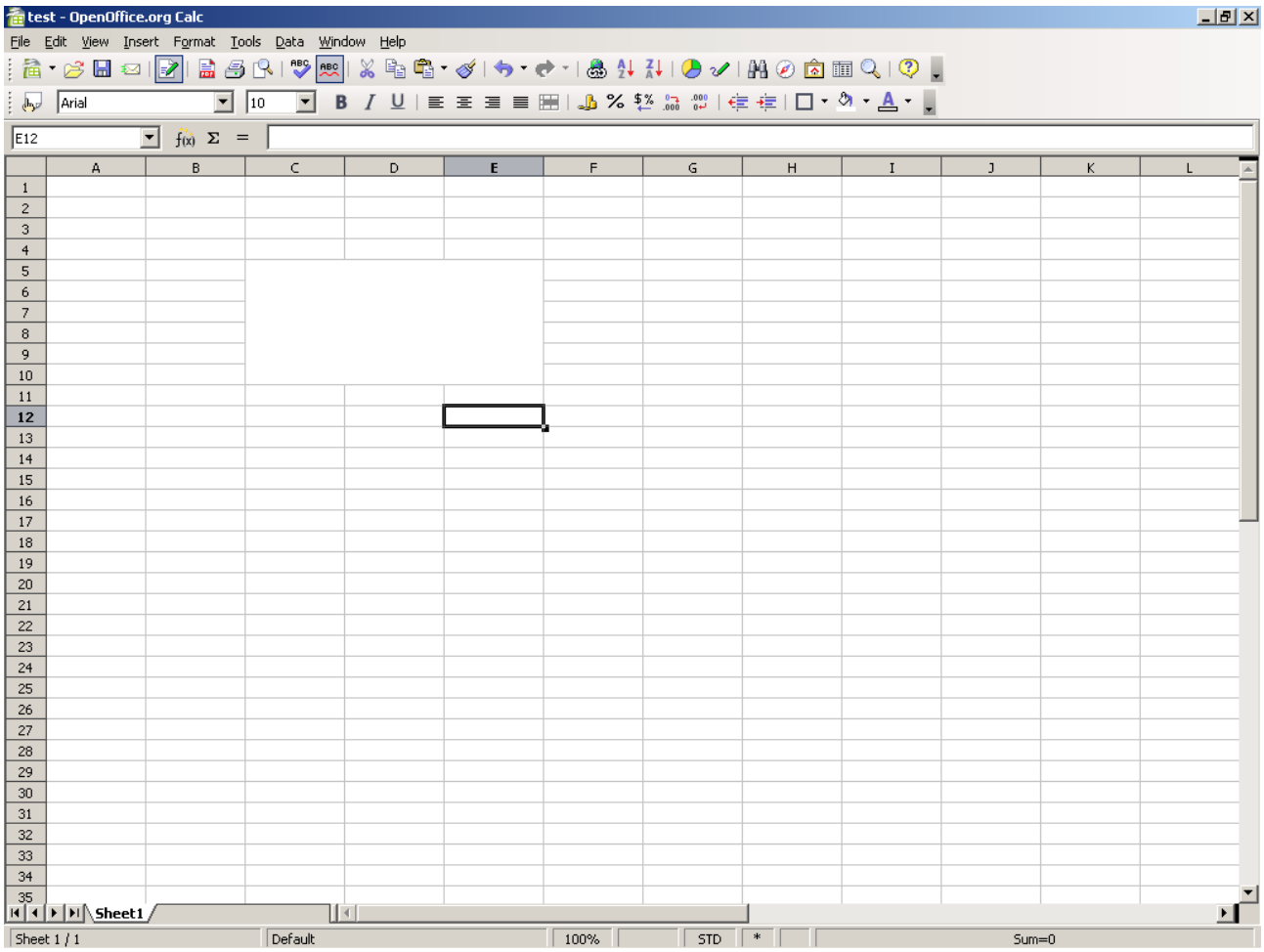

Figure 53: Merged cells.

7 Roundup and Outlook

Open Office provides a good package of programs for offices and constitutes a real alternative to commercial office software solutions. The Microsoft Office ajar user interface simplifies the switch away from the most popular commercial solution. Open Office contains many functions which can be found at commercial office packages, but its value relies in its additional features which have been proven to be very useful.

Therefore the development of additional macros is very important to Open Office, because these macros serve as extensions to single programs or the whole environment and force the development of Open Office. Thus the writings of Mr. Pitonyak, regarding macro programming of Open Office, provide indispensable value when it comes to solve different problems writing a macro.

Using ooRexx to automate Open Office is not only an easy way to write a macro, this also grants a more precise view behind the scenes of Open Office, than writing a macro using Star Basic. This includes the services and interfaces, that are hidden from the user, when using Star Basic. Of course, the BSF4Rexx Library also provides functions, which ease handling Open Office, and therefore hide the complex steps, needed to fulfill these tasks. One example would be the establishment of a connection by using "`uno.connect()`" function, as mentioned in this paper.

Another advantage of the usage of ooRexx is, that programming macros with ooRexx also allows to show, how macros can be started inside, as well as outside of Open Office. This paper also tried to show, how such macros differ from each other and at which point these differences disappear. The result of this is a common header of all macros described in this work.

Unfortunately there are still problems when running macros:

- One of these problems is, that using Microsoft Windows native file or folder dialogs, the initial path is not set.
- Another problem appears when running a macro by pressing a tool bar button. After some successful executions Open Office crashes, if the button is pressed again.

- If a previously created tool bar is removed and then created again, the tool bar is empty and cannot be removed again. By closing Open Office the empty tool bar can be removed.
- Also, when deleting a code block, which was imported by "`w_ImportCode.rex`" and calling this macro at the same cursor position again, some colors are different.

Maybe these problems will be solved in one of the following versions of Open Office.

But this paper is far away from being complete and there is still more work to do. In this paper the user defined dialogs and its components are only described basically. So one task for a future work might be, to reveal the possibilities of creating more complex dialogs.

Another challenge, which has not been covered in this paper, is to define tool bars within the Open Office addons. By using the addressing of macros inside such addons, which is provided by this paper, one should be able to create tool bars connected to the macros provided in the same addon. When installing the addon, the tool bar will be created automatically.

8 Sources

Bibliography

[BSFInstall01] Rony G. Flatscher: Readme of BSF4Rexx, Chapter(s) Installation in a "Ten Second" Nutshell, file:///BSF4Rexx_Install/readmeBSF4Rexx.txt, retrieved on 21.07.08

[GETBSF] Download BSF4Rexx, http://wi-wu-wien.ac.at/rgf/rexx/bsf4rexx/current/, retrieved on 16.08.2008

[GETREXX] ooRexx Download, http://www.oorexx.org/download.html, retrieved on 16.08.2008

[IDLRef01] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/script/provider/XScriptContext.html, retrieved on 22.05.08

[IDLRef02] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/uno/XComponentContext.html, retrieved on 22.05.08

[IDLRef03] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/Desktop.html, retrieved on 22.05.08

[IDLRef04] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XDesktop.html, retrieved on 22.05.08

[IDLRef05] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/document/OfficeDocument.html, retrieved on 22.05.08

[IDLRef06] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/xml/dom/XDocument.html, retrieved on 22.05.08

[IDLRef07] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/MultiServiceFactory.html, retrieved on 22.05.08

[IDLRef08] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/XMultiServiceFactory.html, retrieved on 22.05.08

[IDLRef09] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/DispatchHelper.html, retrieved on 22.05.08

[IDLRef10] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XDispatchHelper.html, retrieved on 22.05.08

[IDLRef100] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XSheetOperation.html, retrieved on 22.05.08

[IDLRef101] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XSpreadsheetView.html, retrieved on 22.05.08

[IDLRef102] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XUsedAreaCursor.html, retrieved on 22.05.08

[IDLRef103] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/Spreadsheet.html, retrieved on 22.05.08

[IDLRef104] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/drawing/XDrawPageSupplier.html, retrieved on 22.05.08

[IDLRef105] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/drawing/XDrawPage.html, retrieved on 22.05.08

[IDLRef106] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/drawing/LineShape.html, retrieved on 22.05.08

[IDLRef107] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/Size.html, retrieved on 22.05.08

[IDLRef108] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/Point.html, retrieved on 22.05.08

[IDLRef109] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/drawing/XShape.html, retrieved on 22.05.08

[IDLRef11] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XDispatchProvider.html, retrieved on 22.05.08

[IDLRef110] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/drawing/GraphicObjectShape.html, retrieved on 22.05.08

[IDLRef111] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XSpreadsheetDocument.html, retrieved on 22.05.08

[IDLRef112] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XSpreadsheets.html, retrieved on 22.05.08

[IDLRef113] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/SheetCellRanges.html, retrieved on 22.05.08

[IDLRef114] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/view/XSelectionSupplier.html, retrieved on 22.05.08

[IDLRef115] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XCellAddressable.html, retrieved on 22.05.08

[IDLRef116] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/table/CellAddress.html, retrieved on 22.05.08

[IDLRef117] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/textfield/URL.html#URL, retrieved on 22.05.08

[IDLRef118] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XMergeable.html, retrieved on 22.05.08

[IDLRef12] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/beans/PropertyValue.html, retrieved on 22.05.08

[IDLRef13] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/bridge/UnoUrlResolver.html, retrieved on 22.05.08

[IDLRef14] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/bridge/XUnoUrlResolver.html, retrieved on 22.05.08

[IDLRef15] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XModifiable.html, retrieved on 22.05.08

[IDLRef16] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XStorable.html, retrieved on 22.05.08

[IDLRef17] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XModel.html, retrieved on 22.05.08

[IDLRef18] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XCloseable.html, retrieved on 22.05.08

[IDLRef19] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/PathSettings.html, retrieved on 22.05.08

[IDLRef20] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/beans/PropertySet.html, retrieved on 22.05.08

[IDLRef21] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/beans/XPropertySet.html, retrieved on 22.05.08

[IDLRef22] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/dialogs/FolderPicker.html, retrieved on 22.05.08

[IDLRef23] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/dialogs/XFolderPicker.html, retrieved on 22.05.08

[IDLRef24] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/ModuleUIConfigurationManagerSupplier.html, retrieved on 22.05.08

[IDLRef25] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/XModuleUIConfigurationManagerSupplier.html, retrieved on 22.05.08

[IDLRef26] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/XUIConfigurationManager.html, retrieved on 22.05.08

[IDLRef27] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/container/XIndexContainer.html, retrieved on 22.05.08

[IDLRef28] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/ItemType.html, retrieved on 22.05.08

[IDLRef29] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/datatransfer/clipboard/SystemClipboard.html, retrieved on 22.05.08

[IDLRef30] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/datatransfer/clipboard/XClipboard.html, retrieved on 22.05.08

[IDLRef31] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/datatransfer/XTransferable.html, retrieved on 22.05.08

[IDLRef32] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/datatransfer/DataFlavor.html, retrieved on 22.05.08

[IDLRef33] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/script/Converter.html, retrieved on 22.05.08

[IDLRef34] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/script/XTypeConverter.html, retrieved on 22.05.08

[IDLRef35] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/uno/TypeClass.html, retrieved on 22.05.08

[IDLRef36] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextDocument.html, retrieved on 22.05.08

[IDLRef37] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/Text.html, retrieved on 22.05.08

[IDLRef38] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XText.html, retrieved on 22.05.08

[IDLRef39] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/Controller.html, retrieved on 22.05.08

[IDLRef40] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XController.html, retrieved on 22.05.08

[IDLRef41] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextViewCursorSupplier.html, retrieved on 22.05.08

[IDLRef42] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextViewCursor.html, retrieved on 22.05.08

[IDLRef43] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XSimpleText.html, retrieved on 22.05.08

[IDLRef44] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/configuration/ConfigurationProvider.html, retrieved on 22.05.08

[IDLRef45] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/configuration/ConfigurationAccess.html, retrieved on 22.05.08

[IDLRef46] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/container/XNameAccess.html, retrieved on 22.05.08

[IDLRef47] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/UnoControlDialogModel.html, retrieved on 22.05.08

[IDLRef48] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/container/XNameContainer.html, retrieved on 22.05.08

[IDLRef49] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XControl.html, retrieved on 22.05.08

[IDLRef50] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XControlModel.html, retrieved on 22.05.08

[IDLRef51] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XControlContainer.html, retrieved on 22.05.08

[IDLRef52] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/Toolkit.html, retrieved on 22.05.08

[IDLRef53] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XToolkit.html, retrieved on 22.05.08

[IDLRef54] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/Frame.html, retrieved on 22.05.08

[IDLRef55] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/frame/XFrame.html, retrieved on 22.05.08

[IDLRef56] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XWindow.html, retrieved on 22.05.08

[IDLRef57] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/UnoControlFixedTextModel.html, retrieved on 22.05.08

[IDLRef58] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/UnoControlButtonModel.html, retrieved on 22.05.08

[IDLRef59] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XButton.html, retrieved on 22.05.08

[IDLRef60] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/SystemPointer.html, retrieved on 22.05.08

[IDLRef61] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/awt/XPointer.html, retrieved on 22.05.08

[IDLRef62] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/XComponent.html, retrieved on 22.05.08

[IDLRef63] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/PathSubstitution.html, retrieved on 22.05.08

[IDLRef64] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XStringSubstitution.html, retrieved on 22.05.08

[IDLRef65] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ucb/SimpleFileAccess.html, retrieved on 22.05.08

[IDLRef66] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ucb/XSimpleFileAccess.html, retrieved on 22.05.08

[IDLRef67] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/packages/Package.html, retrieved on 22.05.08

[IDLRef68] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/XInitialization.html, retrieved on 22.05.08

[IDLRef69] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/container/XHierarchicalNameAccess.html, retrieved on 22.05.08

[IDLRef70] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/XSingleServiceFactory.html, retrieved on 22.05.08

[IDLRef71] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/XActiveDataSink.html, retrieved on 22.05.08

[IDLRef72] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/Pipe.html, retrieved on 22.05.08

[IDLRef73] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/XOutputStream.html, retrieved on 22.05.08

[IDLRef74] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/XInputStream.html, retrieved on 22.05.08

[IDLRef75] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/TextOutputStream.html, retrieved on 22.05.08

[IDLRef76] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/XActiveDataSource.html, retrieved on 22.05.08

[IDLRef77] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/io/XTextOutputStream.html, retrieved on 22.05.08

[IDLRef78] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XChangesBatch.html, retrieved on 22.05.08

[IDLRef79] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/style/XStyleFamiliesSupplier.html, retrieved on 22.05.08

[IDLRef80] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/style/ParagraphStyle.html, retrieved on 22.05.08

[IDLRef81] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/table/BorderLine.html, retrieved on 22.05.08

[IDLRef82] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/dialogs/FilePicker.html, retrieved on 22.05.08

[IDLRef83] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/dialogs/XFilePicker.html, retrieved on 22.05.08

[IDLRef84] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/ui/dialogs/XFilterManager.html, retrieved on 22.05.08

[IDLRef85] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/document/XDocumentInsertable.html, retrieved on 22.05.08

[IDLRef86] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/textfield/DateTime.html, retrieved on 22.05.08

[IDLRef87] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/XNumberFormatsSupplier.html, retrieved on 22.05.08

[IDLRef88] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/lang/Locale.html, retrieved on 22.05.08

[IDLRef89] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextContent.html, retrieved on 22.05.08

[IDLRef90] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XParagraphCursor.html, retrieved on 22.05.08

[IDLRef91] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextGraphicObjectsSupplier.html, retrieved on 22.05.08

[IDLRef92] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/container/XIndexAccess.html, retrieved on 22.05.08

[IDLRef93] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/graphic/GraphicProvider.html, retrieved on 22.05.08

[IDLRef94] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/graphic/XGraphicProvider.html, retrieved on 22.05.08

[IDLRef95] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/graphic/XGraphic.html, retrieved on 22.05.08

[IDLRef96] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/XTextRange.html, retrieved on 22.05.08

[IDLRef97] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/text/textfield/Annotation.html, retrieved on 22.05.08

[IDLRef98] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/util/Date.html, retrieved on 22.05.08

[IDLRef99] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/common/ref/com/sun/star/sheet/XCalculatable.html, retrieved on 22.05.08

[Jakarta01] The Apache Jakarta Project, http://jakarta.apache.org/bsf/, retrieved on 25.07.08

[JAVAREF01] OpenOffice.org 2.4 API reference, file:///OpenOffice.org_2.4_SDK/docs/java/ref/com/sun/star/comp/helper/Bootstrap.html, retrieved on 22.05.08

[OOFFICE] Open Office download, http://download.openoffice.org/other.html, retrieved on 16.08.2008

[OOoInstall01] Readme of BSF4Rexx for Open Office Installation,Chapter(s) Installation, file:///BSF4Rexx_Install/readmeOOo.txt, retrieved on 21.07.08

[ooRexx01] About Open Object Rexx, http://www.oorexx.org/about.html, retrieved on 07.04.08

[ProfUNO01] Professional UNO,Chapter(s) 3.1, http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.xhtml#1_1_Introduction, retrieved on 03.04.08

[ProfUNO02] Professional UNO,Chapter(s) 3.2.1, http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.xhtml#1_2_1_3_Interfaces, retrieved on 03.04.08

[ProfUNO03] Professional UNO,Chapter(s) 3.3.1, http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.xhtml#1_3_1_2_Importing_a_UNO_Object, retrieved on 03.04.08

[ProfUNO03a] Professional UNO,Chapter(s) 3.3.1, http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.xhtml#1_3_1_1_Starting_OpenOffice.org_in_Listening_Mode, retrieved on 03.04.08

[ProfUNO04] Professional UNO,Chapter(s) 3.4.1, http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.xhtml#1_4_1_Java_Language_Binding, retrieved on 03.04.08

[REFBSF01] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Class BSF.DIALOG, Page(s) 2, , retrieved on 21.07.08

[REFBSF02] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Class BSF, Page(s) 2, , retrieved on 21.07.08

[REFBSF03] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Routines – 3, Page(s) 2, , retrieved on 21.07.08

[REFBSF04] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Routines – 1, Page(s) 2, , retrieved on 21.07.08

[REFBSF05] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Class BSF, Page(s) 2, , retrieved on 21.07.08

[REFBSF06] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Class BSF, Page(s) 2, , retrieved on 21.07.08

[REFBSF07] Rony G. Flatscher: Reference Card BSF.CLS, Chapter(s) Public Class BSF, Page(s) 2, , retrieved on 21.07.08

[REFOOO01] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 19, Page(s) 1, , retrieved on 21.07.08

[REFOOO02] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 6, Page(s) 1, , retrieved on 21.07.08

[REFOOO03] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 9, Page(s) 1, , retrieved on 21.07.08

[REFOOO04] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Environment Object .UNO (A Directory Object), Page(s) 2, , retrieved on 21.07.08

[REFOOO05] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Environment Object .UNO (A Directory Object), Page(s) 2, , retrieved on 21.07.08

[REFOOO06] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 15, Page(s) 1, , retrieved on 21.07.08

[REFOOO07] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 4, Page(s) 1, , retrieved on 21.07.08

[REFOOO08] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 30, Page(s) 1, , retrieved on 21.07.08

[REFOOO09] Rony G. Flatscher: Reference Card UNO.CLS, Chapter(s) Public Routines – 7, Page(s) 1, , retrieved on 21.07.08

[RunMacro01] Documentation/OOoAuthors User Manual/Getting Started/How to run a macro, http://wiki.services.openoffice.org/wiki/Documentation/OOoAuthors_User_Manual/Getting_Started/How_to_run_a_macro, retrieved on 25.07.08

[UMIFOO] Useful Macro Information for Open Office, http://www.pitonyak.org/Andrew-Macro.odt, retrieved on 31.05.2008

[UMIFOO01] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.29.2, Page(s) 86, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO02] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.32.1, Page(s) 98-99, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO03] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.32, Page(s) 98, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO04] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.8, Page(s) 33-34, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO05] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.32, Page(s) 101-102, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO06] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.44.1.1, Page(s) 112-114, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO07] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.32.5, Page(s) 68, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO08] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.7, Page(s) 32-33, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO09] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.31.2, Page(s) 96-98, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO10] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.13, Page(s) 185, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO11] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.12, Page(s) 184, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO12] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.1.1, Page(s) 180-181, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO13] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.24, Page(s) 69-70, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO14] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.16.5, Page(s) 200-201, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO15] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.3.2, Page(s) 154, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO16] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.10, Page(s) 183, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO17] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.7.2, Page(s) 181, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO18] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 7.3.8, Page(s) 162-168, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO19] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 6.21, Page(s) 137, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO20] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 8.5.1, Page(s) 225, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO21] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.30.3, Page(s) 93, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO22] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.30.1, Page(s) 91-92, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO23] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.18.5, Page(s) 60, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO24] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 6.25, Page(s) 142, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[UMIFOO25] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 10.4, Page(s) 268-269, http://www.pitonyak.org/AndrewMacro.odt, retrieved on 31.05.08

[URP01] Jörg Budischewski, Stephan Bergmann, Kai Sommerfeld: UNO Remote Protocol Specification , http://udk.openoffice.org/common/man/spec/urp.html, retrieved on 06.04.08

[VIMONLINE] Vim Download, http://www.vim.org/download.php, retrieved on 16.08.2008

Illustration Index

Sourcecode Index