“AUTOMATING OPENOFFICE.ORG WITH OOREXX: OOREXX NUTSHELL EXAMPLES FOR WRITE AND CALC”

Rony G. Flatscher

Wirtschaftsuniversität Wien (WU), Austria

„The 2005 International Rexx Symposium“, Los Angeles, California, U.S.A.,

April 17th - April 21st, 2005.

ABSTRACT

The opensource Microsoft Office clone "OpenOffice.org" is available on multiple platforms, from Windows, over Linux to OS/2. It can read/write Microsoft office file-formats, such as Word, Excel or PowerPoint. Its scripting architecture is radically different from what Microsoft has come up with and appears to be more systematic, although there is a rather steep learning curve to it.

This article will give numerous little "nutshell" examples of driving OpenOffice.org via ooRexx. All the examples will run unaltered under Linux and Windows.

Keywords: Object Rexx, ooRexx, BSF, BSF4Rexx, Automation, Scripting, OpenOffice.org, OOo.

Disclaimer:

The supporting infrastructures and modules,
namely “BSF4Rexx” and “OOo.CLS”,
introduced and used in this article may change in the future due to their “work in progress” status at the time of this writing!

1 INTRODUCTION

“In the 90ies a German company named "Star Division" developed a portable C++ class library that it named "Star". It became available for MacOS, Unix, OS/2 and Windows. With this infrastructure the company started to develop an integrated office suite which should be portable to all platforms that "Star" was available. It was named "StarOffice".

Due to the success of the Microsoft office (MSO) suite, it was very important for Star Division to create their office suite in a way, which allowed co-operating with MSO users as seemlessly as possible. For that reason powerful import and export filters for MSO have been of paramount importance and in effect allows it today, to import and export most MSO documents into/from StarOffice on any platform Star Office is available.

In addition the StarOffice user interface resembles the MSO user interface, such that it becomes rather feasible for companies to migrate from MSO to StarOffice should they wish to escape a possibly undersired lock-in situation, or should they strategically plan to not be locked-in anymore. Among other benefits such a migration also opens up the possibility to switch operating and hardware system platforms as long as StarOffice is available on the new target.

Finally, the built-in scripting language ("Star Basic") is a look-alike language to Microsoft's Visual Basic, sharing most of the language constructs and properties, thereby making it relatively easy for MSO (end-user) programmers to become productive in StarOffice.

The company Sun, looking for a MSO compatible, powerful office suite for its Unix-based operating system Solaris bought the German company and made it available as a commercial product on Windows as well. In addition, two important developments have taken place since then:

- "StarOffice" has become fully programmable from Java, which is because of Sun's interest to leverage its programming language Java as much as possible,
- Sun released a version of "StarOffice" into the opensource and made it freely available under the name "OpenOffice". The WWW homepage is located at "OpenOffice.org" and it has therefore become common to abbreviate it as "OOo".

Having an opensource version of StarOffice freely available allows companies and organizations to escape potentially undesired lock-ins by switching to the open-source codebase, either immediately or whenever such a company decides this to be a move in its own interest. In the case of switching from StarOffice to OpenOffice (or the other way round) incurs no switching costs whatsoever. If switching from MSO to StarOffice/OOo switching costs (installation, re-training users) need to be taken into account, but it becomes possible to save by cancelling costful upgrade plans in the long run, and possibly remarkable money by being able to switch away from the Windows operating system altogether. Such switching to opensource software that is free, seems to be quite attractive for public administrations in Europe (e.g. German cities, or the Austrian capital Vienna).

Possibly more important, free software allows schools, Universities, third world countries, private people to employ a powerful, integrated office package for their needs, without cutting into their (most of the time extremely tight) budgets. Having the software not only free, but also in opensource means that maintenance of such software is even possible, if the original writers are not available anymore. Especially in the scholar environment opensource allows analysis, improvements and extensions to such software, because of such a white box approach."[1]

This article introduces the results of analyzing the ooRexx nutshell examples for OOo as developed by a student of the author (Augustin, cf. [Aug05]) over the wintersemester 2004/05 at the WU (Wirtschaftsuniversität Wien), creating a requirable ooRexx module which should ease the interaction specifically with UNO components, of which OOo is composed of. It then introduces some nutshell examples relating to the word processor ("swrite") and spreadsheet ("scalc") in a rewritten form, taking advantage of the developed ooRexx module.

[1] Cf. [Flat05], section „Introduction".

2 CREATING AN OOREXX MODULE "OOO.CLS"

This section introduces the ooRexx module "OOo.CLS" which should help ooRexx programmers who wish to program OpenOffice.org (OOo) via its constituent UNO (Universal Network Object) components.[2]

2.1 Analyzing ooRexx Code to Interact with UNO Components

Analyzing the ooRexx examples in [Aug05][3], which are based on information from the OOo developer documentation and information related to scripting in [W3OOoAPI], [W3OOoFW] and [W3OOoUDK]. In researching OOo scripts it is clear that Walter Augustin was looking into Java examples, and trying to rewrite them for ooRexx taking advantage of BSF4Rexx (cf. e.g. [Flat04]).

Figure 1[4] depicts an ooRexx script from [Aug05] which starts up OOo and creates an empty word processor document. Various sections of that code have been highlighted to draw the attention of the reader to the following observations:

1. Most of the program is concerned in creating the runtime environment for driving the UNO components, ending at the statement starting with "oComponentLoader =…".

   All of the examples in [Aug05] possess these statements, which therefore are candidates to be "outsourced" into an own module (package) and made available as a routine.

2. The UNO service objects (instances of UNO components) are queried for interfaces that encapsulate functionality (methods) and/or properties (attributes) belonging semantically together, like "Naming" services, "Factory" services and the like.

   Querying an interface always involves an instance of the Java class "com.sun.star.uno.UnoRuntime", because of its method "queryInterface()", the

---

2 Cf. [Flat05] for an architectural overview of OOo, UNO and how ooRexx can interact taking advantage of the Java UNO bindings, by employing BSF4Rexx (cf. [Flat01], [Flat03], [Flat04]).

3 [Aug05] contains Object REXX examples and their counterparts in JavaScript (Rhino) as well.

4 The program in figure 1 uses a URL to connect to the OOo on the local host, port 8,100. If you have not configured your OOo to listen on that port, and wish to test that program, then you need to start an instance of OOo by issuing the following command in a command line window (cf. [OOo03]):

```
soffice -accept=socket,host=localhost,port=8100;urp;
```

Java class object of the interface one queries (first argument to supply) and the service object of which the interface is sought for (second argument to supply).

It is noticeable that “queryInterface()” is used quite often (five times in figure 1 alone) and always consumes two statements, one to get at the Java class object representing the desired interface, and one for invoking the queryInterface() run-time method itself.

It may be interesting to note that OOo interface names always start out with a capital “X” (an OOo convention).

3. The class “com.sun.star.beans.PropertyValue” is of utmost importance to OOo programs. Many times additional information, attributes for services etc. are denoted with Java arrays of this type.

   A “PropertyValue” consists of four fields only[5], the two most important being: “Name” and “Value”, where “Name” is a string and “Value” an instance of the Java class of the appropriate type[6].

   It is also interesting to note that in the case of creating empty documents, one needs to supply an empty array object of type “PropertyValue”.

[5] Classes that consist of fields (aka “attributes”, “properties”) only, are called “structs”.

[6] The primitive Java types “boolean”, “byte”, “char”, “short”, “int”, “long”, “float”, “double” must be represented as instances of the classes “java.lang.Boolean”, “java.lang.Byte”, “java.lang.Character”, “java.lang.Short”, “java.lang.Integer”, “java.lang.Long”, “java.lang.Float”, “java.lang.Double”, respectively. So an ooRexx function that “box”es the primitive datatypes to their Java class counterparts (and an “unbox” routine for the opposite purpose) would be desirable in the future.

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
sComponentContext = .bsf~new("com.sun.star.comp.helper.Bootstrap") ~createInitialComponentContext(.nil)
unoRuntime = .bsf~new("com.sun.star.uno.UnoRuntime")

sUrlResolver = sComponentContext~getServiceManager() ~createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver", sComponentContext)
XUnoUrlResolver = .bsf4rexx~Class.class~forName("com.sun.star.bridge.XUnoUrlResolver")
oUrlResolver = unoRuntime~queryInterface(XUnoUrlResolver, sUrlResolver)

unoUrl = "uno:socket,host=localhost,port=8100;urp;StarOffice.NamingService"
oInitialObject = oUrlResolver~resolve(unoUrl)
XNamingService = .bsf4rexx~Class.class~forName("com.sun.star.uno.XNamingService")
sNamingService = unoRuntime~queryInterface(XNamingService, oInitialObject)

oServiceManager = sNamingService~getRegisteredObject("StarOffice.ServiceManager")
XMSFactory = .bsf4rexx~Class.class~forName("com.sun.star.lang.XMultiServiceFactory")
sMSFactory = unoRuntime~queryInterface(XMSFactory, oServiceManager)

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
sDesktop = sMSFactory~createInstance("com.sun.star.frame.Desktop")
XDesktop = .bsf4rexx~Class.class~forName("com.sun.star.frame.XDesktop")
oDesktop = unoRuntime~queryInterface(XDesktop, sDesktop)
XComponentLoaderName = .bsf4rexx~Class.class~forName("com.sun.star.frame.XComponentLoader")
oComponentLoader = unoRuntime~queryInterface(XComponentLoaderName, oDesktop)

/* Open a blank text document   */
/* No properties needed         */
propertyValueName = .bsf4rexx~Class.class~forName("com.sun.star.beans.PropertyValue")
loadProps = .bsf~createArray(propertyValueName, 0) /* 0=no elements, i.e. empty Java array */
/* load an empty text document */
sWriterComponent = oComponentLoader~loadComponentFromURL("private:factory/swriter", "_blank", 0, loadProps)
::requires "BSF.cls"
```

Figure 1: A Typical ooRexx OOo Program (Modeled After Java Programs).

2.2 Devising a Prototype ooRexx OOo-Module

In this section a prototype ooRexx module will get developed which should ease programming in ooRexx for OOo. The module's name should be "OOO.CLS" (OpenOffice.org class) and should supply at least the following features:

1. Contain the initialisation code for the UNO runtime environment,
2. allow to easily create the OOo desktop object, which is the pivotal object for creating and interacting with the OOo components,
3. make it considerably easier to query (and get) interface objects compared to Java,
4. make it easier loading OOo classes.

The module will require "BSF.CLS", the support for bridging ooRexx with Java using the BSF4Rexx (cf. [Flat04]) package. This way ooRexx is able to use UNO components from ooRexx via their Java interfaces.

In this section most of the module gets introduced and explained, the entire module is depicted in appendix A at the end of this article.

2.2.1 Initialisation of the ooRexx Module "OOO.CLS"

In the initialization part of this module a directory object should be created and stored in the ".local" ooRexx runtime environment. This directory object should serve as a directory maintaining runtime information that is important for OOo and consequently be named with the ooRexx environment symbol[7] ".OOO".

One important application of this ".OOO" directory is its purpose to maintain the most important UNO/OOo objects and make them easily accessible.

Firstly, the Java class objects[8] for the following important UNO/OOo classes should be stored with their unqualified name in the ".OOO" directory:

- "com.sun.star.beans.PropertyValue": needed for supplying properties to UNO/OOo components,

[7] In ooRexx an identifier that starts with a dot is called "environment symbol": the runtime system will try to lookup an entry in one of its runtime environments by removing the leading dot and using the rest of the (uppercased) identifier as the key.

[8] Using BSF4Rexx (cf. [Flat04]) allows to interact with those Java class objects as if they were ooRexx objects, which makes sending ooRexx messages to them straight forward, and still will cause the invocation of the respective Java methods.

- "com.sun.star.comp.helper.Bootstrap": needed for bootstrapping UNO/OOo,
- "com.sun.star.uno.UnoRuntime": needed mostly for querying the interface objects.

Secondly, the names of those UNO/OOo interfaces should be stored with the help of the ".OOO" directory object which are needed by the ooRexx nutshell programs. For this purpose a directory object is created and stored with the name "XInterfaces" in ".OOO". Each entry in the XInterfaces directory is pointing to the fully qualified name of the interface, using the fully qualified name itself and with a second entry, using the unqualified name as an index.[9]

2.2.2 The ooRexx Class "OOO_PROXY"

The ooRexx class "OOO_PROXY" allows to wrap up any BSF object and add OOo specific behaviour, namely the dealing with messages that indeed are meant for querying interface objects. The class "OOO_PROXY" is depicted in figure 2. The ooRexx constructor method "INIT" merely saves the BSF object in an attribute (ooRexx "object variable", which can be directly accessed from each method via the "EXPOSE" statement issued as the very first statement).

The behavior to query interface objects is added by exploiting the ooRexx "UNKNOWN"-mechanism: whenever a message is sent to an ooRexx object for which no method can be found, the "NOMETHOD" exception is raised by the ooRexx runtime system which interrupts and stops the execution of an ooRexx program. If a class which was sought for a method which has the same name as the received message name contains a method by the name of "UNKNOWN", then the ooRexx runtime system will invoke this method instead, supplying as the first argument the name of the message for which no method was found and a second argument, either being ".nil" or an array object containing the arguments which were supplied to the message in round parenthesis.

The "OOO_PROXY" class implements an "UNKNOWN" method to intercept all messages for which no methods could be found. If the message starts with a capital "X", then it is assumed that the programmer wishes to query the interface object from the

[9] This way an interface has always two entries in the "XInterfaces" directory, e.g. "com.sun.star.uno.XComponentContext" will be stored with an index name of "COM.SUN.STAR.UNO.XCOMPONENTCONTEXT" and an index name of "XCOMPONENTCONTEXT".

```
/* ********************************************************
   Class:       OOo_PROXY
   Purpose:     allows to wrap up BSF (Java) objects as UNO/OOo objects;
                if instances are destroyed, then no derigistration from
                the BSF registry takes place
   ******************************************************** */
::CLASS OOo_PROXY

::Method bsfObject ATTRIBUTE   -- stores the BSF object

::METHOD init           -- constructor method
   expose bsfObject    -- direct access to BSF object
   use arg bsfObject   -- assign object to attribute
   self~objectName=bsfObject~objectName -- use same objectname as BSF proxy

::METHOD unknown
   expose bsfObject    -- direct access to BSF object
   parse arg xName

   if xName~left(1)~translate="X" then -- if an interface object is asked for, query & return it
   do
      xClass=.OOo~entry(xName)   -- try to get interface class object
      if xClass=.nil then        -- no entry as of yet
      do
         idx=substr(xName, lastpos(".", xName)+1)
         xClass=OOo.loadClass(.ooo~XInterfaces~entry(xName), idx)
      end

         -- query and return the interface object, wrap it up as an OOO_PROXY
      return .OOo_proxy~new(.OOo~unoRuntime~queryInterface(xClass, bsfObject))
   end
   else  -- pass to BSF object (i.e. the BSF class) to handle it
   do
      if bsfObject~hasmethod(xName) then    -- o.k. it's a BSF method, forward it
         FORWARD TO (bsfObject) MESSAGE (xName) ARGUMENTS (arg(2))
      else
         FORWARD TO (bsfObject) CONTINUE    -- let BSF handle this

      tmpObj=result  -- retrieve the returned object

      -- if a BSF-object, wrap it up as an OOO_PROXY (to gain this UNKNOWN behaviour)
      if tmpObj~hasmethod("bsf.setFieldValue") then return .OOo_proxy~new(tmpObj)
                                               else return tmpObj
   end
```

Figure 2: "OOO.CLS": The "OOO_PROXY" Class.

receiving object. In this case, the entries in the directory object "XInterfaces" as stored in ".OOO" are inspected to learn about the fully qualified name of the interface, which then is used to query the interface object. In order to optimize the runtime behavior the Java class objects representing the desired interfaces are created and stored in the ".OOO" directory by their (unqualified) class name.[10] In any case the received interface object will be wrapped up as an instance of the "OOO_PROXY" class, such that it receives that particular "UNKNOWN" behavior as well.

If the unknown message does not start with a capital "X", then it is either a message which should invoke a BSF method or a message which mostlikely should invoke a

[10] In a subsequent run the stored Java class object will be reused.

Java method with the help of BSF. In the case of a BSF object the received interface object will be wrapped up as an instance of the “OOO_PROXY” class, such that it receives that particular “UNKNOWN” behavior as well. Otherwise the received string value will be returned.

2.2.3 Public Routine “OOo.loadClass”

The public routine “OOo.loadClass” expects two arguments, the fully qualified name of the (Java) class to load, and optionally a string determining the index value to use to store the Java class object in the “.OOO” directory object. If the second argument is omitted, then the (unqualified) class name is used as the index value.

The fully qualified Java class name is used to load the Java class object with the help of BSF and gets wrapped up as an “OOO_PROXY” object. Figure 3 depicts the ooRexx code.

```
     -- load the given OOo-class and save it in environment
::routine OOo.loadClass public
     parse arg className, idx

     if idx="" then   -- omitted, hence extract last word to serve as index
        idx=substr(className, lastpos(".", className)+1)

     if .OOo~hasentry(idx)=.true then -- idx already used (class maybe loaded already)
        .error~say("OOo.loadClass:" pp(idx) "already used, in hand:" -
                   pp(className) "in .OOo:" pp(.OOo~entry(idx)))

        -- load Java class via BSF, wrap it up as an "OOO_PROXY"
     tmpClass=.OOo_proxy~new(.bsf4rexx~class.class~forName(className))
     .OOo~setentry(idx, tmpClass)   -- save it in the ".OOo" directory
     return tmpClass                -- return the wrapped class object
```

Figure 3: “OOO.CLS”: Public Routine “OOO.loadClass”.

2.2.4 Public Routine: “OOO.connect”

Figure 4 depicts the public ooRexx routine “OOO.CONNECT”[11] which initializes the UNO runtime environment. The strategy employed allows for using URL-style connections to OOo server instances, running anywhere on the Internet. It is assumed that the local installation of OOo got configured such, that it is addressable via the TCP/IP port number 8,100 as per the OOo developer guide (cf. [OOo03]).[12]

[11] The Rexx language allows the dot to be part of an identifier (as well as the question mark, exclamation mark and the underline character, besides letters and digits).

[12] An alternative would be to use instead the method „bootstrap()“ from ooRexx which is defined in

```
      -- connects to given UNO server, returns the XMultiServiceFactory
      -- interface object of the remote service manager
::routine OOo.connect public
  parse arg unoURL
      -- no unoURL given, use default
  if unoURL="" then unoURL="uno:socket,host=localhost,port=8100;urp;StarOffice.NamingService"

  context=.OOo~Bootstrap~createInitialComponentContext(.nil)
  uur= context~getServiceManager() –
                ~createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver", context)

  remoteObject        = uur~XUnoUrlResolver ~resolve(unoURL)
  remoteNamingService = remoteObject~XNamingService

  remoteServiceManager=remoteNamingService~getRegisteredObject("StarOffice.ServiceManager")
  remoteMSF           = remoteServiceManager~XMultiServiceFactory

  if .OOo~hasentry("remoteMSF")=.false then
     .OOo~remoteMSF=remoteMSF -- save remote MultiServiceFactory

  return remoteMSF  -- return the remote XMultiServiceFactory interface object
```

Figure 4: "OOO.CLS": Public Routine "OOO.connect".

This routine will return a "XMultiServiceFactory" interface object which can be used to create the OOo desktop object and more. For later referral this object will get stored in the ".OOO" directory under the name "remoteMSF".

2.2.5 Public Routine: "OOO.getDesktop"

This routine creates a desktop service object, queries its desktop interface object and returns it. If no context object is supplied as an argument, then the one stored in the ".OOO" directory is used. The code of this routine is depicted in figure 5.

```
   -- create and return the OOo Desktop interface object
::routine OOo.getDesktop
  USE ARG remoteMSF

  if remoteMSF="" | remoteMSF=.nil then -- no vaild argument supplied
     remoteMSF=.OOo~remoteMSF           -- use the one stored by "OOO.connect"

      -- create the OOo Desktop service object
  remoteDesktop=remoteMSF~createInstance("com.sun.star.frame.Desktop")

      -- query and return its desktop interface object
  return .OOo~unoRuntime~queryInterface(.OOo~XDesktop, remoteDesktop)
```

Figure 5: "OOO.CLS": Public Routine "OOO.getDesktop".

---

the class "com.sun.star.comp.helper.Bootstrap", e.g.:
`xContext = .bsf~bsf.import("com.sun.star.comp.helper.Bootstrap")~bootstrap`

2.2.6 Putting the Module “OOO.CLS” to Work

Figure 6 shows the ooRexx code that is needed for creating an empty instance of a word processor component, if using the module “OOO.CLS”. This has exactly the same effect, as the Java transcription to ooRexx of figure 1, which was the basis of the analysis and the reason for devising the module “OOO.CLS”. Therefore one can expect that some code can be saved, nevertheless it is noticeable how much of the original code can be left out.

Taking advantage of the module allows the resulting code to become considerably more concise, and by the same token it also becomes easier to read and to understand.

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMSFactory=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
sDesktop         = xMSFactory~createInstance("com.sun.star.frame.Desktop")
oDesktop         = sDesktop~XDesktop          -- get desktop interface
oComponentLoader = oDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
sWriterComponent = oComponentLoader~loadComponentFromURL("private:factory/swriter", "_blank", 0, .OOo~noProps)

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

Figure 6: Program of Figure 1 Rewritten, Using the Module “OOO.CLS”.

3 NUTSHELL EXAMPLES

The examples in this chapter originate in the work of [Aug05], who created them following the Java examples available in [OOo03], [W3OOoFW] and on the Internet in general. The ooRexx examples could be simplified quite considerably in this article by taking advantage of the ooRexx module “OOO.CLS”, making it easier to understand and extend on them.

In order to run the examples one needs to have Object REXX or its opensource version “Open Object Rexx” (ooRexx, cf. [W3ooRexx]) and BSF4Rexx (cf. [W3B4R]) installed. In addition the Java classpath needs to point to the OOo Java archives according to the documentation in [OOo03] and the ooRexx module “OOO.CLS” from the appendix of this article needs to be accessible[13] by these ooRexx programs.

As described in section 2.2.4, the public routine “OOO.connect” of the ooRexx module “OOO.CLS” uses the explicit port number 8,100, so one needs to set up OOo accordingly. Alternatively, one can invoke an instance of OOo via the commandline, instructing it to accept client connections via port 8,100. This can be done by issuing the following command in a commandline window:[14]

```
soffice -accept=socket,host=localhost,port=8100;urp;
```

Most of the OOo nutshell examples of this chapter run *unchanged* on Windows and Linux.[15] For each nutshell example a brief description is given.

3.1 “swriter” Nutshell Example # 1

Figure 7 depicts an ooRexx nutshell example that creates an empty word processor component and saves that empty document in a Microsoft Word format, such that MS Word users are able to work with that document. This nutshell relates to [Aug05] “Example_08.rex”.

To store a document one needs to acquire the “XStorable” interface object and use

[13] Any Rexx program/module is “accessible” if it resides in the same directory as the running Rexx program or in a directory denominated in the operating system environment variable “PATH”.

[14] A future implementation of the routine „OOO.connect” may forgo this need by using the “bootstrap()” method of the UNO Bootstrap class instead. If so, the ability of being able to connect to an OOo server instance via TCP/IP should be preserved.

[15] In this implementation of the ooRexx module „OOO.CLS“ fully qualified filenames need to be adjusted to the target platform as the Windows version always contains a “drive letter” (a letter followed immediately by a colon).

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
xWriterComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/swriter", "_blank", 0, .OOo~noProps)

-- Get a "Storable" interface in the context of the Writer application
xStorable=xWriterComponent~XStorable

-- This time we need to define a property: The filter name
storeProps = .bsf~createArray(.OOo~propertyValue, 1)
storeProps[1] = .bsf~new("com.sun.star.beans.PropertyValue")
storeProps[1]~bsf.setFieldValue("Name", "FilterName")
storeProps[1]~bsf.setFieldValue("Value", "MS Word 97")

storeUrl = "file:///c:/test.doc"      -- Linux/UNIX users change the filename!
xStorable~storeAsURL(storeUrl, storeProps)

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

Figure 7: "swriter" Nutshell Example # 1 (Saving as a MS Word Document).

its methods to store it. In this case the method "storeAsURL" is used, which expects as arguments the document's file name in URI form and an array of properties which give additional information about how the storing should be carried out. In this example one property, "FilterName", is given with a value of "Word 97", causing OOo to store the document in MS Word 97 format.

3.2 "swriter" Nutshell Example # 2

Figure 8 depicts an ooRexx nutshell example that creates an empty word processor component and prints that empty document to a locally installed printer. This nutshell relates to [Aug05] "Example_10.rex".

To print a document one needs to acquire the "XPrintable" interface object and use its methods to set the printer and to print the document. In this case the method "setPrinter" is used to determine the printer which should get used by supplying its name in its argument (an array that contains one property element). The method "print" is used to invoke the print process, which gets the information via the property array to print the first page only!

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
xWriterComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/swriter", "_blank", 0, .OOo~noProps)

-- Get a "Printable" interface in the context of the Writer application
xPrintable=xWriterComponent~XPrintable

-- We need to define a property: The printer name
props = .bsf~createArray(.OOo~propertyValue, 1)
props[1] = .bsf~new("com.sun.star.beans.PropertyValue")

-- This is a printer's name as known in your system.

props[1]~bsf.setFieldValue("Name", "Name")
props[1]~bsf.setFieldValue("Value", "Apple Color LW 12/660 PS on file")
xPrintable~setPrinter(props)

-- This prints the (empty) test page
props[1]~bsf.setFieldValue("Name", "Pages")
props[1]~bsf.setFieldValue("Value", "1")
xPrintable~print(props);

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

Figure 8: "swriter" Nutshell Example # 2 (Print the First Page of a Document).

3.3 "swriter" Nutshell Example # 3

Figure 9 depicts an ooRexx nutshell example that creates an empty word processor component, changes the style of a paragraph, inserts the current page number into it and finally adds some text at the end to it. This nutshell relates to [Aug05] "Example_24.rex".

This example demonstrates the Model-View-Controller (MVC) paradigm that OOo implements: in order to manipulate the content of a document one needs to get the controller object of the document which can then be used to get the "view cursor". A "cursor" allows addressing certain parts and types of information in a document, in our example we use a "view cursor" from which a "page cursor" is requested.

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document

oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

-- ---------- End of connection header ---------------

/* load an empty text document */
oTextComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/swriter", "_blank", 0, .OOo~noProps)

xDocument=oTextComponent~XTextDocument
xText=xDocument~getText

-- Now we could start writing into the document, for example: xText~setString("A few words ...")

-- In order to set paragraph properties, we need a cursor for navigation.
-- A cursor can be retrieved from the controller, the controller comes from the model,
-- and the model comes from the component, i.e. the application.


xModel=xDocument~XModel                      -- Get model from component

xController = xModel~getCurrentController    -- The model knows its controller

       -- The controller gives us the TextViewCursor, query the ViewCursor supplier interface
xViewCursorSupplier=xController~XTextViewCursorSupplier

xViewCursor = xViewCursorSupplier~getViewCursor      -- Get the ViewCursor

       -- Set the appropriate property for paragraph style
xCursorPropertySet=xViewCursor~XPropertySet

xCursorPropertySet~bsf.invokeStrict("setPropertyValue", "STRING", "ParaStyleName", -
       "STRING", "Quotations")

       -- Print the current page number – we need the XPageCursor interface for this
xPageCursor=xViewCursor~XPageCursor

       -- This is written directly into the open document (see above).
xText~setString("The current page number is:" xPageCursor~getPage)

       -- The text creates a model cursor from the viewcursor
xModelCursor = xText~createTextCursorByRange(xViewCursor~getStart)

       -- Now we could query XWordCursor, XSentenceCursor and XParagraphCursor
       -- or XDocumentInsertable, XSortable or XContentEnumerationAccess
       -- and work with the properties of com.sun.star.text.TextCursor

       -- In this case we just go to the end of the paragraph and add some text.
       -- Now get a paragraph cursor first.
xParagraphCursor=xModelCursor~XParagraphCursor

       -- Go to the end of the paragraph
xParagraphCursor~gotoEndOfParagraph(.false)
xParagraphCursor~setString(" ***** Fin de semana! ******")


::requires OOo.cls
```

Figure 9: "swriter" Nutshell Example # 3 (Edit Text in Word Processor Document).

3.4 “scalc” Nutshell Example # 1

Figure 10 depicts an ooRexx nutshell example that creates an empty spreadsheet document and adds two entries into it. This nutshell relates to [Aug05] “Example_14.rex”.

From the code it can be seen that a spreadsheet document can be regarded as a collection of sheets. Therefore it is important to address that collection and to pick the sheet one wishes to interact with; in this case a “XIndexAccess” interface object is created which allows to index its collection with an integer number (index starts with 0). All co-ordinates are 0-based, hence the cell “A1” (left top cell) resides in “column=0” and “row=0”.

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
oSheetComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/scalc", "_blank", 0, .OOo~noProps)

xDocument=oSheetComponent~XSpreadSheetDocument
xSheets=xDocument~getSheets
xIndexSheets=xSheets~XIndexAccess
xSheet=xIndexSheets~getByIndex(0)~XSpreadSheet  -- get the spreadsheet interface for the sheet

-- (1) insert a value (number)
call setCell 1, 1, 123.456, xSheet, 1
-- (2) insert a text string (formula, text)
call setCell 1, 2, "Hello", xSheet, 0


::requires OOo.cls    -- get OOo support

-- A function for inserting values or formulas into cells
-- Parameters:
--      x, y           cell coordinates
--      content        value or formula to insert
--      container      the spreadsheet or cell range where the cell resides
--      isValue        (1 [true] or 0) indicates whether the content should be
--                     regarded as a number or a character string
::routine setCell
   use arg x, y, content, container, isValue

        oCell = container~getCellByPosition(x, y)
        if isValue then
                oCell~setValue(content)
        else
                oCell~setFormula(content)
        RETURN
```

Figure 10: “scalc” Nutshell Example # 1 (Enter Values into a Spreadsheet).

3.5 “scalc” Nutshell Example # 2

Figure 11 depicts an ooRexx nutshell example that creates an empty spreadsheet document and adds two entries into it. This nutshell relates to [Aug05] “Example_18.rex”.

The program first fills in a few values using the routine “setCell”, calculates the average of a given area in routine “avgNonEmpty” and stores it in the cell with the coordination “0, 3” (i.e. “column=0”, “row=3”; or: “A4”).

3.6 “scalc” Nutshell Example # 3

Figure 12 depicts an ooRexx nutshell example that creates an empty spreadsheet document, adds entries to it and uses them to create a 3-D diagram. This nutshell relates to [Aug05] “Example_22.rex”.

The program first fills in a few values, then creates a rectangle which will contain the diagram created from the values and finally will turn the shape of the diagram from 2-D to 3-D.

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
oSheetComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/scalc", "_blank", 0, .OOo~noProps)

xDocument=oSheetComponent~XSpreadSheetDocument
xSheets=xDocument~getSheets
xIndexSheets=xSheets~XIndexAccess
xSheet=xIndexSheets~getByIndex(0)~XSpreadSheet  -- get the spreadsheet interface for the sheet

-- (1) setCell dummy data
call setCell 0, 0, 5.2, xSheet, 1
call setCell 0, 2, 2.3, xSheet, 1

-- (2) Calculate and print average
result = avgNonEmpty(0, 0, 0, 2, xSheet)
call setCell 0, 3, result, xSheet, 1

-- (3) setCell some labels
call setCell 1, 1, "<- Remains empty", xSheet, 0
call setCell 1, 3, "<- The average", xSheet, 0


::requires OOo.cls    -- get OOo support

-- A function for inserting values or formulas into cells
-- Parameters:
--      x, y           cell coordinates
--      content        value or formula to insert
--      container      the spreadsheet or cell range where the cell resides
--      isValue        (1 [true] or 0) indicates whether the content should be
--                     regarded as a number or a character string
::routine setCell
   use arg x, y, content, container, isValue

        oCell = container~getCellByPosition(x, y)
        if isValue then oCell~setValue(content)
                   else oCell~setFormula(content)
        return

-- A function to calculate the average not considering empty fields
::routine avgNonEmpty
use arg fromx, fromy, tox, toy, container
        sum = 0
        fieldCount = 0

        do i=fromx to tox
                do j=fromy to toy
                        currentValue = container~getCellByPosition(i, j) ~getValue
                        if currentValue <> 0 then
                        do
                                sum = sum + currentValue
                                fieldCount = fieldCount + 1
                        end
                end
        end

        if fieldCount > 0 then return (sum/fieldCount)
        return 0
```

Figure 11: “scalc” Nutshell Example # 2 (Enter and Retrieve Spreadsheet Values).

```
/* initialize connection to server, get its Desktop-service and XComponentLoader interface */
xMsf=ooo.connect()   -- connect to server and retrieve remote multi server factory

-- Retrieve the Desktop object, we need its XComponentLoader interface
-- to load a new document
oDesktop         = xMsf~createInstance("com.sun.star.frame.Desktop")
xDesktop         = oDesktop~XDesktop          -- get desktop interface
xComponentLoader = xDesktop~XComponentLoader  -- get componentLoader interface

/* load an empty text document */
oSheetComponent = xComponentLoader~loadComponentFromURL( -
        "private:factory/scalc", "_blank", 0, .OOo~noProps)

xDocument=oSheetComponent~XSpreadSheetDocument
xSheets=xDocument~getSheets
xIndexSheets=xSheets~XIndexAccess
xSheet=xIndexSheets~getByIndex(0)~XSpreadSheet  -- get the spreadsheet interface for the sheet

-- As a prerequisite, we populate the spreadsheet with some numeric values:
do i=0 to 4
       do j=0 to 3
              call setCell i, j, i+j, xSheet, 1
       end
end

-- First create the elements the chart consists of:
-- (1) a frame (made from the Rectangle class)
oRect = .bsf~new("com.sun.star.awt.Rectangle")
oRect~bsf.setFieldValue("X", 500)
oRect~bsf.setFieldValue("Y", 3000)
oRect~bsf.setFieldValue("Width", 25000)
oRect~bsf.setFieldValue("Height", 11000)

-- (2) the underlying data (the numeric values come from a cell range)
myRange=xSheet~XCellRange ~getCellRangeByName("A1:E4")

-- Get the address of the underlying data (in our case a cell range)
myAddr=myRange~XCellRangeAddressable ~getRangeAddress

cellRangeAddressName = .bsf4rexx~Class.class~forName("com.sun.star.table.CellRangeAddress")
oAddr = .bsf~createArray(cellRangeAddressName, 1)
oAddr[1] = myAddr

-- Now get the chart collection from the Sheet's charts supplier and add a new chart
-- Get the supplier and its charts
oCharts=xSheet~XTableChartsSupplier ~getCharts

-- Append a new chart to the collection
oCharts~addNewByName("Example", oRect, oAddr, "true", "true")

-- change diagram properties: 2-D to 3-D
oChart=oCharts~XNameAccess ~getByName("Example")
xChart=oChart~XTableChart
oEmb=xChart~XEmbeddedObjectSupplier ~getEmbeddedObject
oDiag=oEmb~XChartDocument ~getDiagram
oDiag~XPropertySet ~setPropertyValue("Dim3D", .bsf~new("java.lang.Boolean", "true"))

::requires OOo.cls    -- get OOo support

::routine setCell
   use arg x, y, content, container, isValue

       oCell = container~getCellByPosition(x, y)
       if isValue then
              oCell~setValue(content)
       else
              oCell~setFormula(content)
       return
```

Figure 12: “scalc” Nutshell Example # 3 (Enter Spreadsheet Values, Create Diagram).

4 SUMMARY AND OUTLOOK

This article introduced the reader to developing and employing a new ooRexx module, “OOo.cls”, which is intended at easying the programming of OpenOffice.org (OOo) from ooRexx in a portable manner. As a starting point the ooRexx nutshell examples modeled after Java and JavaScript in [Aug05] served as a “study object” to find out code sequences that are either repetitive (e.g. the initialization of UNO components) or which state a recurrent pattern (e.g. the need for carrying out queryInterface-invocations on UNO service objects).

The effects of employing “OOo.cls” were demonstrated by rewriting an original [Aug05] example (cf. figure 1), which was modeled after Java examples given by OOo code repository sites (e.g. [W3OOoAPI]), yielding the ooRexx program as depicted in figure 6.

After this proof-of-concept a few nutshell examples which demonstrate the interaction with the word processor (“swriter”) and the spreadsheet (“scalc”) component of OOo are given. These nutshells are intended as starting points to help the reader understand the necessary steps to take in order to be able to drive OOo.

As the support is realized via the Java interfaces of the UNO components (cf. [OOo03]) which constitute OOo (cf. [Flat05]), it is necessary to use the BSF4Rexx (cf. [Flat04]) package, which allows ooRexx to almost transparently address Java.

This BSF4Rexx support can be further improved, e.g. by supplying “box” and “unbox” routines that will create from primitive datatypes Java objects instantiating their appropriate Java class and vice-versa. Also, being able to query and set Java fields by merely sending the field names to the Java object proxies (as is the case with ooRexx object attributes) would ease programming quite a bit.

Finally, the support for the interface classes in “OOo.cls” needs to be replenished. It may be possible – upon further research – to take advantage of Java and UNO reflection to transparently handle the querying of interface classes dynamically at runtime.

5 REFERENCES

[Aug05] Augustin A.: “Examples for Open Office Automation with Scripting Languages”, Bachelor course paper, Wirtschaftsuniversität Wien, January 2005.

[Ende97] Ender T.: "Object-Oriented Programming with REXX", John Wiley & Sons, New York et.al. 1997.

[Flat01] Flatscher R.G.: "Java Bean Scripting with Rexx", in: Proceedings of the „12th International Rexx Symposium“, Raleigh, North Carolina, USA, April 30th - May 2nd, 2001.

[Flat03] Flatscher R.G.: “The Augsburg Version of BSF4Rexx”, in: Proceedings of the „14th International Rexx Symposium“, Raleigh, North Carolina, USA, May 5th - May 7th, 2003.

[Flat04] Flatscher R.G.: “Camouflaging Java as Object REXX”, in: Proceedings of the „2004 International Rexx Symposium“, Böblingen, Germany, May 3rd - May 6th, 2004.

[Flat05] Flatscher R.G.: “Automating OpenOffice with ooRexx: Architecture, Gluing to Rexx Using BSF4Rexx”, in: Proceedings of the „The 2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 17th - April 21st, 2005.

[OOo03] N.N.: “OpenOffice.org 1.1 - Developer’s Guide”, PDF download from [W3OOo], June 2003.

[Pit04] Pitonyak A.: “OpenOffice.org Macros Explained”, Hetzenwerke Publishing, Whitefish Bay 2004.

[VeTrUr96] Veneskey G., Trosky W., Urbaniak J.: "Object Rexx by Example", Aviar, Pittsburgh 1996.

[W3BSF] Homepage of Apaches “Bean Scripting Framework” (BSF), URL (2005-05-17): `http://jakarta.apache.org/bsf`

[W3B4R] Beta test and release candidate site of the “BSF4Rexx” package, URL(2004-05-01): `http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/`

[W3ObjRexx] Open Object Rexx homepage, URL (2005-05-17):

http://www.ooRexx.org

[W3OOo] OpenOffice.org homepage, URL (2005-05-17): http://www.OpenOffice.org/

[W3OOoAPI] OpenOffice.org API Homepage, URL (2005-05-17): http://api.OpenOffice.org/

[W3OOoFW] OpenOffice.org Framework Homepage with Links to Scripting, URL (2005-05-17): http://framework.openoffice.org/

[W3OOoUDK] OpenOffice.org UDK (UNO development kit) homepage, URL (2005-05-17): http://udk.OpenOffice.org/

[W3ooRexx] Homepage of “Open Object Rexx” (ooRexx), URL (2005-05-17): http://www.ooRexx.org

[W3RexxLA] Homepage of the “Rexx Language Association”, URL (2005-05-17): http://www.RexxLA.org

APPENDIX A

Full listing of the ooRexx module “OOo.CLS”.

```
/*
   author:  Rony G. Flatscher
   date:    2005-04-18/19/20
   version: 0.0.3
   name:    OOo.cls
   purpose: ease interfacing with OpenOffice.org, v1.1.x and up
   needs:   BSF4Rexx

*/


   -- if no environment entry, then create the entry with default classes
if .local~hasentry("OOo")=.false then
do
   .local~OOo=.directory~new  -- create a new directory object and store it in .local

   -- preload classes
   l = .list~of(                                          -
      "com.sun.star.beans.PropertyValue"                , -
      "com.sun.star.comp.helper.Bootstrap"              , -
      "com.sun.star.uno.AnyConverter"                   , -
      "com.sun.star.uno.UnoRuntime"                     -
      )

   do c over l -- get Java class object and save it in .OOo
      call OOo.loadClass c
   end

   .OOo~noProps=.bsf~createArray(.OOo~propertyValue, 0)   -- create empty array of property-value

   -- list of Interface classes to be loaded at runtime (needs completion)
   l = .list~of(                                          -
      "com.sun.star.beans.XIntrospectionAccess"         , -
      "com.sun.star.beans.XMultiPropertySet"            , -
      "com.sun.star.beans.XPropertySet"                 , -
      "com.sun.star.bridge.XUnoUrlResolver"             , -
      "com.sun.star.container.XEnumeration"             , -
      "com.sun.star.container.XIndexAccess"             , -
      "com.sun.star.container.XIndexContainer"          , -
      "com.sun.star.container.XNameAccess"              , -
      "com.sun.star.container.XNameContainer"           , -
      "com.sun.star.container.XNamed"                   , -
      "com.sun.star.chart.XChartDocument"               , -
      "com.sun.star.document.XEmbeddedObjectSupplier"   , -
      "com.sun.star.drawing.XDrawPage"                  , -
      "com.sun.star.drawing.XDrawPagesSupplier"         , -
      "com.sun.star.drawing.XShape"                     , -
      "com.sun.star.drawing.XShapes"                    , -
      "com.sun.star.frame.XComponentLoader"             , -
      "com.sun.star.frame.XController"                  , -
      "com.sun.star.frame.XDesktop"                     , -
      "com.sun.star.frame.XFrame"                       , -
      "com.sun.star.frame.XFramesSupplier"              , -
      "com.sun.star.frame.XModel"                       , -
      "com.sun.star.frame.XStorable"                    , -
      "com.sun.star.lang.XComponent"                    , -
      "com.sun.star.lang.XMultiComponentFactory"        , -
      "com.sun.star.lang.XMultiServiceFactory"          , -
      "com.sun.star.lang.XServiceInfo"                  , -
      "com.sun.star.lang.XSingleServiceFactory"         , -
      "com.sun.star.lang.XTypeProvider"                 , -
      "com.sun.star.presentation.XPresentationSupplier" , -
      "com.sun.star.script.XTypeConverter"              , -
      "com.sun.star.sheet.XCellRangeAddressable"        , -
      "com.sun.star.sheet.XCellRangesQuery"             , -
      "com.sun.star.sheet.XSpreadsheet"                 , -
      "com.sun.star.sheet.XSpreadsheetDocument"         , -
      "com.sun.star.sheet.XSpreadsheetView"             , -
      "com.sun.star.sheet.XSpreadsheets"                , -
      "com.sun.star.style.XStyleFamiliesSupplier"       , -
```

```
      "com.sun.star.table.XCell"                          , -
      "com.sun.star.table.XCellRange"                     , -
      "com.sun.star.table.XTableChart"                    , -
      "com.sun.star.table.XTableChartsSupplier"           , -
      "com.sun.star.text.XPageCursor"                     , -
      "com.sun.star.text.XParagraphCursor"                , -
      "com.sun.star.text.XSentenceCursor"                 , -
      "com.sun.star.text.XText"                           , -
      "com.sun.star.text.XTextContent"                    , -
      "com.sun.star.text.XTextCursor"                     , -
      "com.sun.star.text.XTextDocument"                   , -
      "com.sun.star.text.XTextRange"                      , -
      "com.sun.star.text.XTextViewCursorSupplier"         , -
      "com.sun.star.text.XWordCursor"                     , -
      "com.sun.star.uno.XComponentContext"                , -
      "com.sun.star.uno.XNamingService"                   , -
      "com.sun.star.uno.XInterface"                       , -
      "com.sun.star.util.XModifiable"                     , -
      "com.sun.star.util.XRefreshable"                    , -
      "com.sun.star.util.XSearchable"                     , -
      "com.sun.star.view.XPrintable"                        -
      )

   -- save interface class names with which we want to deal
   d=.directory~new
   do i over l
     idx=substr(i, lastpos(".", i)+1)
     d~setentry(i, i)    -- by fully qualified name
     d~setentry(idx, i)  -- by last word
   end
   .OOo~XInterfaces = d  -- save directory
end


::requires "BSF.cls"    -- get BSF4Rexx support


      -- load OOo-class and save it in environment
::routine OOo.loadClass public
     parse arg className, idx

     if idx="" then idx=substr(className, lastpos(".", className)+1) -- extract last word to serve as index
     if .OOo~hasentry(idx)=.true then    -- e.g. last word is used as a class in a another package as well
        .error~say("OOo.loadClass:" pp(idx) "already used, in hand:" pp(className) "in .OOo:" pp(.OOo~entry(idx)

-- say "className="className "idx="idx
     tmpClass=.OOo_proxy~new(.bsf4rexx~class.class~forName(className))  -- wrap it up
     .OOo~setentry(idx, tmpClass)   -- load the given Java class and save it in .OOo
     return tmpClass



      -- connects to given UNO server, returns the XMultiServiceFactory interface object of the remote service m
::routine OOo.connect public
  parse arg unoURL
      -- no unoURL given, use default
  if unoURL="" then unoURL="uno:socket,host=localhost,port=8100;urp;StarOffice.NamingService"

  context=.OOo~Bootstrap~createInitialComponentContext(.nil)
  uur= context~getServiceManager() ~createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver", context)

  remoteObject        = uur~XUnoUrlResolver ~resolve(unoURL)
  remoteNamingService = remoteObject~XNamingService

  remoteServiceManager=remoteNamingService~getRegisteredObject("StarOffice.ServiceManager")
  remoteMSF           = remoteServiceManager~XMultiServiceFactory
  if .OOo~hasentry("remoteMSF")=.false then .OOo~remoteMSF=remoteMSF -- save remote MultiServiceFactory
  return remoteMSF  -- return the remote XMultiServiceFactory
```

```
      -- get and return the XDesktop interface object
::routine OOo.getDesktop
  USE ARG remoteMSF

  if remoteMSF="" | remoteMSF=.nil then remoteMSF=.OOo~remoteMSF -- use the stored remote MultiServiceFactory
  remoteDesktop=remoteMSF~createInstance("com.sun.star.frame.Desktop")
      -- return the XDesktop interface of the remote desktop
  return .OOo~unoRuntime~queryInterface(.OOo~XDesktop, remoteDesktop)



::routine iif public
  if arg(1)=.true then return arg(2)
                  else return arg(3)



/* ***********************************************************
   Class:       OOo_PROXY
   Purpose:     allows to wrap up Java objects as Object Rexx objects;
                if instances are destroyed, then no derigstration from
                the BSF registry takes place
   restriction: needed to make getting interface objects easier
   *********************************************************** */
::CLASS OOo_PROXY -- subclass BSF PUBLIC

::Method bsfObject ATTRIBUTE

::METHOD init
   expose arg bsfObject -- direct access to BSF object
   use arg bsfObject    -- assign object to attribute
   self~objectName=bsfObject~objectName   -- use same objectname as BSF proxy (= idx into BSF-registry)

::METHOD unknown
   expose arg bsfObject -- direct access to BSF object
   parse arg xName

   if xName~left(1)~translate="X" then -- if an interface object is asked for, query and return it
   do
      xClass=.OOo~entry(xName)-- try to get interface class object
      if xClass=.nil then        -- no entry as of yet
      do
         idx=substr(xName, lastpos(".", xName)+1)
         xClass=OOo.loadClass(.ooo~XInterfaces~entry(xName), idx)
      end

         -- query and return the interface object
      return .OOo_proxy~new(.OOo~unoRuntime~queryInterface(xClass, bsfObject))
   end
   else  -- pass to super class (i.e. BSF) to handle it
   do
      if bsfObject~hasmethod(xName) then -- o.k. it's a BSF method
         FORWARD TO (bsfObject) MESSAGE (xName) ARGUMENTS (arg(2))
      else
         FORWARD TO (bsfObject) CONTINUE    -- let BSF handle this
      tmpObj=result
      -- if a BSF-object, wrap it up
      if tmpObj~hasmethod("bsf.setFieldValue") then return .OOo_proxy~new(tmpObj)
                                               else return tmpObj
   end
```

Date (latest version) of Article: 2005-05-17.

Published in: Proceedings of the „2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 18th - April 21st, 2005, The Rexx Language Association, Raleigh N.C., 2005.

Presented at: „2005 International Rexx Symposium“, Los Angeles, California, U.S.A., April 20th, 2005.