Wirtschaftsuniversität Wien

Abteilung für Wirtschaftsinformatik

LV-Nr.: 1826 SS 2006
Vertiefungskurs VI: Projektseminar

LV-Leiter: Univ. Prof. Dr. Rony G. Flatscher

Seminararbeit

BSF4Rexx and OpenOffice.org

Nutshell-Examples

Autoren:

Gerhard Görlich, Matr. Nr. 0251857

Åsmund Realfsen, Matr. Nr. 0250879

David Spanberger, Matr. Nr. 0353637

Table of Contents

Illustration Index

1 Abstract

In this paper a series of nutshell examples for BSF4Rexx[1] are presented.

BSF4Rexx is a software that makes it possible for Open Object Rexx (ooRexx) to use Java functions an libraries. In addition BSF4Rexx makes it possible for Java applications to use ooRexx as a scripting engine. Only the first topic is covered in this work.

With BSF4Rexx all kinds of applications that provide a Java programming interface can be accessed and controlled using ooRexx. OpenOffice.org provides a Java API for all its modules, and is because of this controlable from ooRexx with the help of BSF4Rexx.

This work consist of two parts. In the first part the theory of ooRexx, BSF4Rexx and programming within OpenOffice.org is explained. In the second part a series of nutshell examples of how to access and utilize Java from ooRexx is presented.

The examples are divided into three categories.

In the first category (C1) BSF4Rexx is used to connect ooRexx with the standard Java API, and perform some simple operations. In addition some external libraries are used.

In the second category (C2) ooRexx is used to control OpenOffice.org. These examples contain automation of common tasks like printing, opening documents, inserting data into a spreadsheet or create a chart.

In the third category (C3) the knowledge from category one and two is used in order to create a bit more advanced examples.

---

[1]   Bean Scripting Framework for Rexx

2 System-Description

This chapter describes the technologies of ooRexx, BSF4Rexx and OpenOffice.org in a theoretical way and builds the base for the practical examples in chapter 3.

2.1 Bean Scripting Framework

„Bean Scripting Framework (BSF) is a set of Java classes which provides scripting language support within Java applications, and access to Java objects and methods from scripting languages. BSF allows one to write JSPs in languages other than Java while providing access to the Java class library...“ [AJP06]

A Bean Scripting framework enables scriping langauges like Tcl, Python, ooRexx to access Java Classes, Objects and their methods. But it also enables Java to execute programs written in a supported scripting language.

2.1.1 History

The development of BSF started in 1999 at the Watson Research Center. The initial intention was to make Java Beans available from scripting languages, so that they can access the enormous archive of Java components. More and more developers noticed the usefulness of this framework. So the global interest in this technology grew very fast. In 2002 BSF get added to Jakarta Project which is part of the Apache Software Foundation and offers open source Java solutions. Until nowadays, many improvements had been made and BSF has reached its version 2.3. [AJP06]

2.1.2 Architecture
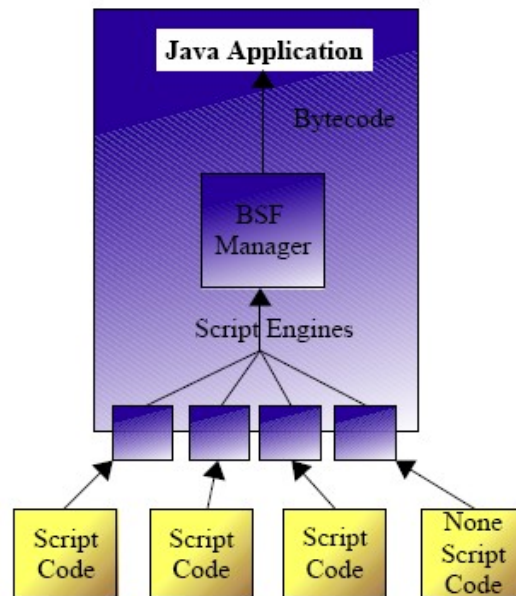
Figure 1: Architectural Overview [Hane05].

There are two major components of BSF:

- *"The `BSFManager` handles all scripting execution engines running under its control, and maintains the object registry that permits scripts access to Java objects. By creating an instance of the `BSFManager` class, a Java application can gain access to scripting services."* [AJP06]

  The BSFManager cares about the scripting execution engines of the supported languages. So by creating an instance of it, Java programs can execute Code for the scripting languages.

  The other way round, the BSFManager handles the object registry of Java and allows scripting languages to access these objects.

- *„The `BSFEngine` provides an interface that must be implemented for a language to be used by BSF. This interface provides an abstraction of the scripting language's capabilities that permits generic handling of script execution and object registration within the execution context of the scripting language engine. "* [AJP06]

  The BSFEngine provides the interfaces, which are doors between Java and the scripting languages. It offers a common interface for all the supported languages. So a Java program can executed code for different scripting languages via the same interface.

2.2 BSF4Rexx

BSF4Rexx is a Bean Scripting Framework for the scripting language ooRexx. With BSF4Rexx Java Classes, Objects and methods can be accessed by ooRexx and ooRexx scripts can be executed by Java. [Burger05]

2.2.1 History

- "Essener Version"

  This version was developed by Prof. Flatscher and his student Peter Kalender in the years 2000/2001. In this version only Java could execute ooRexx code.

- "Augsburger Version"

  This version was developed by Prof. Flatscher in 2003. With this version it was possible to access Java Classes, objects and methods from ooRexx.

- "Vienna Version"

  This is the latest version of BSF4Rexx. Among other improvements this version offers the usage of typeless variables and support for using automating OpenOffice.org.

2.2.2Architecture

The following figure shows the architecture of the "Vienna Version".
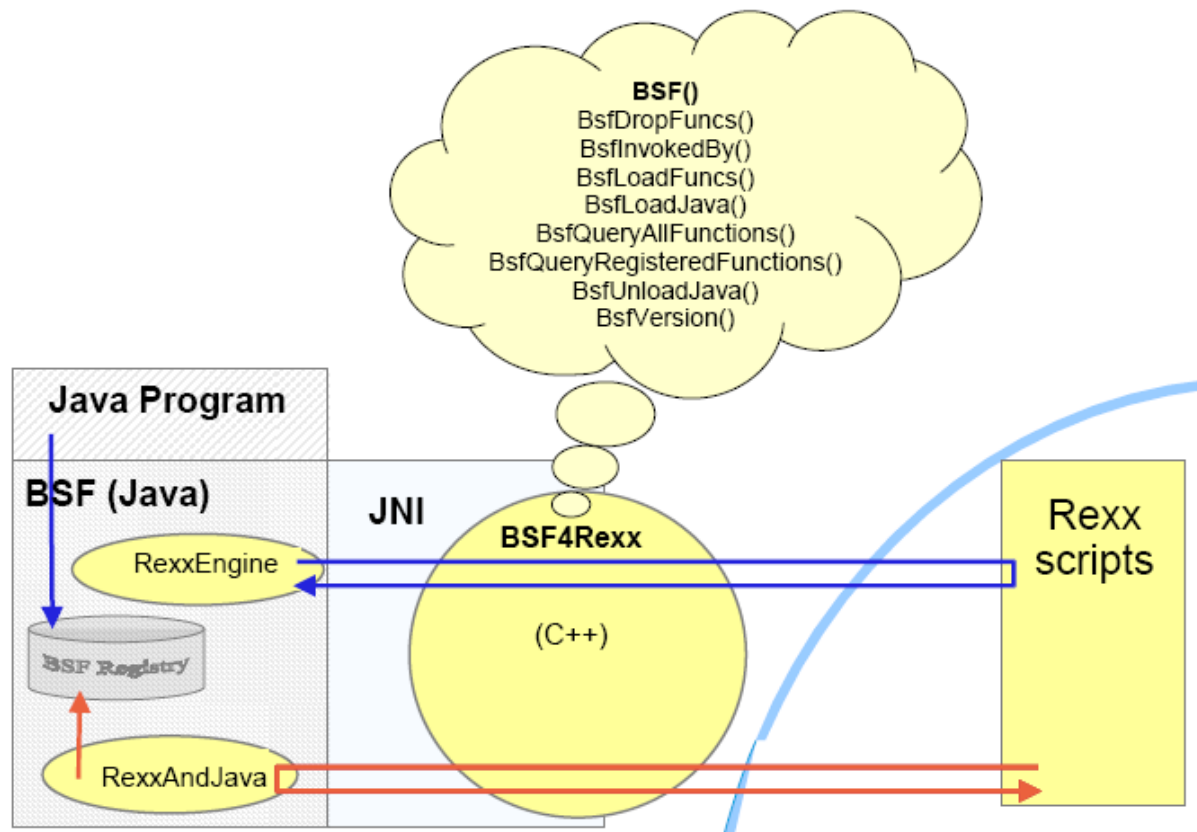
Figure 2: Communication between Rexx and Java with BSF4Rexx [Flat06].

The following example should give a guess how Java can be addressed by ooRexx.

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

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

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

Figure 3: Code Example for connecting to Java [Flat06]

Depending on the version of Java a possible output could be: *Java.version: 1.5.0_06*

The newer versions come together with the file "bsf.cls". It contains several functions, which can save a lot of time. Importing this file in a ooRexx script with the line "::requires bsf.cls" makes this support available.
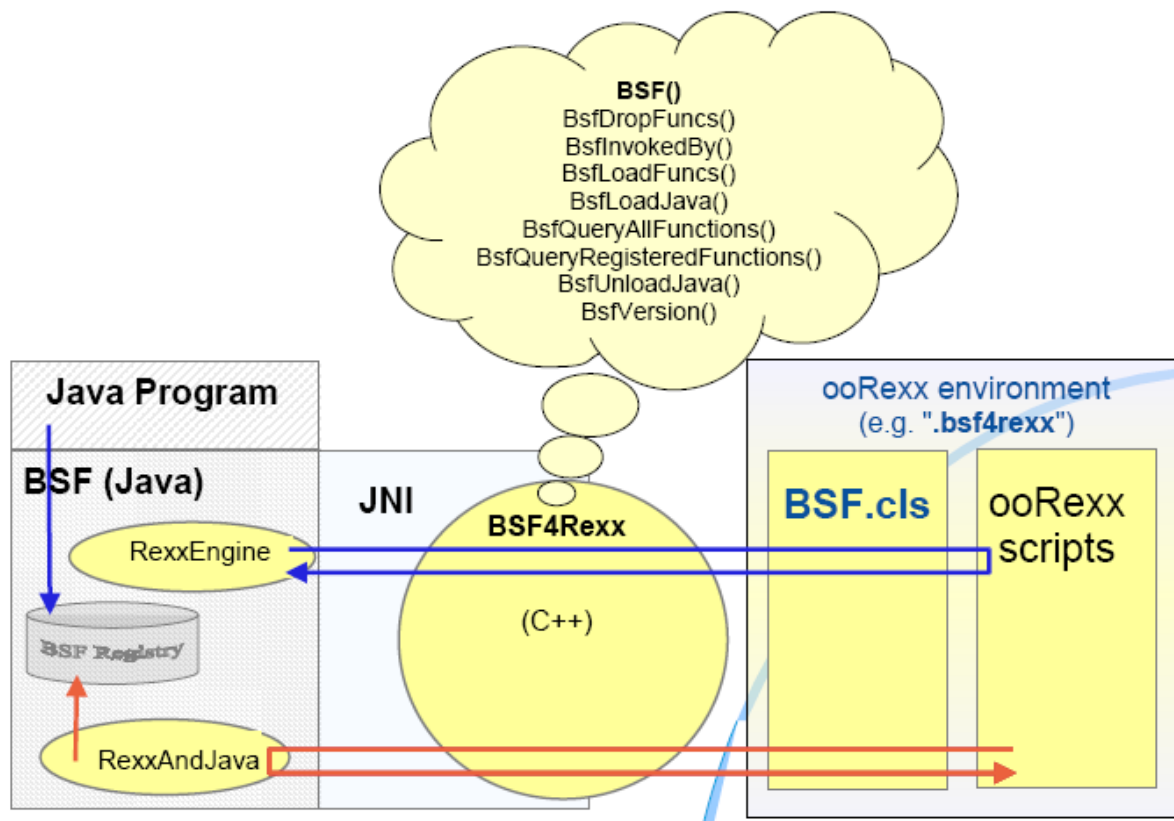
Figure 4: Communication between Rexx and Java using BSF.cls [Flat06]

The following example should show, that in this simple case, the code length can be re-duced by the factor 3.

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

Figure 5: Code example for connecting to Java using BSF.cls [Flat06]

Although the code has only two lines the output, depending on the version of Java, is the same: *Java.version: 1.5.0_06*

2.3 ooRexx

Open Object Rexx (ooRexx) is a object oriented scripting language which is available on a wide range of different platforms. ooRexx has several distinct features:[ooR05]

- A syntax near natural English language,

- easy to use because of few rules,

- interpreted line for line,

- many built-in functions and

- typeless variables.

Additionally programming interfaces exist for many applications and operating systems.

2.3.1 History

The history of ooRexx starts in 1979 when Mike Cowlishaw started the development of the Rexx language. (REstructured eXtended eXecutor) Cowlishaw worked for IBM and and he tried to make a scripting language that could replace the EXEC and EXEC 2 languages. It was designed to be a scripting language that could run on a wide range of systems, and not dependent on one type of technology. Over the years IBM included Rexx as a scripting language in almost all their own operating systems. They also made packages for other famous operating systems like Novell Netware, Windows and Linux. Also several non IBM versions was developed. [Wiki06]

In 1996 American National Standards Institute (ANSI) published an own standard for Rexx, the ANSI X3.274–1996 standard. [Wiki06]

Rexx is a procedural language and it has no support for object oriented concepts. In 2004 IBM released an object oriended version of Rexx : Object Rexx. Object Rexx is the basis for ooRexx that was first released in 2005. [Wiki06] ooRexx is open source software licensed under the Common Public License (CPL) v1.0. The current version of ooRexx is 3.0.0 for Unix and 3.0.1 for Windows.

2.3.2 Syntax and Use

In order to understand the examples provided in this work, a basic knowledge of the Rexx language is necessary. In this chapter some basic Rexx language constructs are explained. The ooRexx reference[2] guide can be consulted, if additional knowledge is required.

2.3.2.1 Variables and Output

```
A = 5
b = 10
say "a + b =" a + b
```

In the code above the number 5 is saved in variable *A*, and 10 is saved in variable *b*. In the third line the command *say* is used in order to write first the text "*a + b =*" on the screen, and then the sum of variable *A* and *b* is added.

ooRexx is case insensitive, meaning the command "*a = 5*" is the same as "*A = 5*". Because of this ooRexx feature, the use of a lower case "a" in third line, and a upper case "A" in the first line does not make any trouble.

Text surrounded by quotation marks is always interpreted as text, and not as a command or a variable.

The code above gives the following output.

```
a + b = 15
```

2.3.2.2 Loops

In ooRexx loops are created with a *DO .... END* statement, that comes in many different variations. One simple variation is demonstrated in the code below.

```
a = 5
do until a = 10
    say "a =" a
    a = a + 1
end
```

The output will be that the script prints all numbers from 5 to 9 on the screen.(inclusive)

---

2   http://www.ooRexx.org/Rexxref.pdf

2.3.2.3 Routines

A routine is simply created with the command ::routine, and end with the return statement.

```
say pow3(2)

::routine pow3
    use arg a
return a * a * a
```

In the code above, the *say* command calls the *pow3* routine with an argument with the value *2*. The *pow3* routine returns the argument multiplied 3 times with is self. The output of the code block above is "*8*".

2.4 OpenOffice.org

OpenOffice.org is an open source office suite developed by the OpenOffice.org project. OpenOffice.org is distributed under the terms of the LGPL (see also C1-11 p. 62). This licensing policy allows the usage of the code for the non open source StarOffice suite, distributed by Sun Microsystems. This commercial OpenOffice.org version is supplemented with some proprietary software components like the "Enterprise Tools" package. The current version is 8.0 [Star06]. OpenOffice.org uses the vendor-neutral OASIS open document format, an open, XML-based standard (ISO 23600) for office documents [OASIS06].

The following Table shows the history and developement of OpenOffice.org:

1984	The German company StarDivision created the first version of the StarOffice suite
1999	StarDivision was bought by Sun Microsystems. Sun offered StarOffice 5.2 as a free download.
2000	Sun Microsystems announced the OpenOffice.org project. In the same year the OpenOffice.org homepage was set up and the source code of StarOffice 6.0 could be downloaded and modified. Proprietary, non open source, software components had been removed by Sun Microsystems.
2001	The first version of OpenOffice.org was released, Build 638c
2002	OpenOffice.org version 1.0 was released
2005	The latest version, 2.0, was created.

2.4.1 Components

The OpenOffice.org office suite provides six major applications: Writer, Impress, Math, Draw, Calc and Base [cp. OO06]:

2.4.1.1 Writer

Writer is a modern word processing tool. It offers a large set of features for editing and formatting documents. Wizards help the user creating standard documents such as letters or faxes. The "AutoComplete" feature suggests common words and phrases to complete

what the user is typing at the moment. A built-in spell checker checks the document on the fly and the thesaurus feature can be used to get synonyms for words (see also example C2-4). Furthermore Writer can import Microsoft Word documents and export documents to HTML or Portable Document Format (.pdf) and has a built in interface to email software.

2.4.1.2 Impress

Impress is a tool for creating multimedia presentations. It offers multiple views at the current presentation such as Slides, Notes, Outline or Handout view. A set of drawing tools allow the creation of diagrams and pictures. Presentations can be enriched by using animation effects or clip arts. Impress can import Microsoft PowerPoint presentations and export presentations to many different Formats such as .pdf, Flash (.swf), pictures (JPEG) or XHTML

2.4.1.3 Math

Math is a component for creating mathematical equitations. It can create well formatted equitations in text documents or formulas for Calc. It can be used as a stand-alone application or with other OpenOffice.org components.

2.4.1.4 Draw

Draw is a tool for creating diagrams. It offers various formatting and style options for 2D and also 3D objects. Draw can import most common graphic formats like JPEG, PNG, GIF and export graphics to Flash (.swf).

2.4.1.5 Calc

Calc is a spreadsheet program. It offers wizards for a wide range of spreadsheet functions and diagrams. It allows to import and process data from external data sources and databases. A "Scenario Manager" allows "what if..." analysis. Calc can import Microsoft Excel documents and export documents among others to .pdf and Microsoft Excel format.

2.4.1.6 Base

Base allows to manipulate databases from OpenOffice.org. Base comes with a built-in HSQL database engine (a database engine written in Java), but can also be used with other database systems. Base enables the user to create reports, forms, SQL views and manipulate data.

2.4.2 OpenOffice.org Versions

Apart from StarOffice there are some other Office suites based on OpenOffice.org, e.g. AOL Office, KaiOffice, MagyarOffice (Hungarian), NextOffice (for Mac) and Pladao Office (Thai). Furthermore there is an OpenOffice.org version for Windows that can be installed on USB sticks (86MB) called Portable OpenOffice.org, created by John Haller [portOO06].

2.4.3 Universal Network Objects (UNO)

UNO is the base component technology of OpenOffice.org. UNO components are objects in the form of compiled and bound libraries. They must implement certain interfaces and run in a UNO context. These objects are specified by the interfaces they implement, described in a language independent interface description language (UNOIDL) which is similar to CORBA or MIDL. Communication between objects takes place only over calls to these interfaces [devel05, p. 65]. This concept allows UNO components to be accessed and implemented in every language for which a UNO language binding exists [udk06]. Currently complete language bindings exist for [udk06]:

- C++

- Java

- Python

Additionally there exists bindings that allow access but not the development of new components. These are bindings for [udk06]:

- OpenOffice.org Basic

- Common Language Infrastructure (CLI) from which Microsoft .NET is an implementation

- Object Linking and Embedding (OLE) Automation

UNO also provides a way of communication between it's components. So called "bridges" can send method calls and receive return values between processes and objects written in different programming languages. These bridges use a UNO specific remote protocol (URP) and can communicate over sockets or pipes [devel05, p. 65]. Figure 1 illustrates the communication of UNO components. Each of the components exists in a UNO

environment which is provided as language binding. E.g. Component one could be written in C++, accessed with the C++ language binding in a Unix context while component two could be written in Java, running in a Java virtual machine on a Windows system:
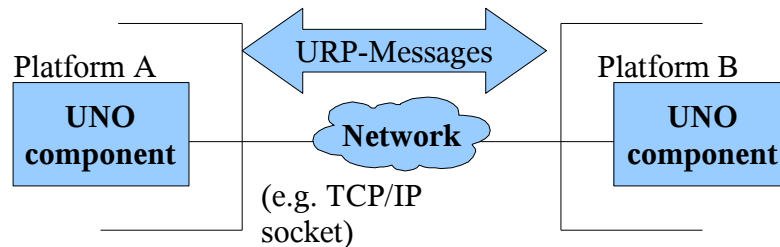
Figure 6: UNO component communication.

Figure 6 shows how the UNO components form different UNO applications[3]. The applications have some specific components (e.g. a text document in the Writer), but they also use common components (e.g. printer settings) or components from other applications (e.g. diagrams in Impress). In this way code can be reused. Each component can be exchanged easily with an other component if it provides the same interfaces which makes the system flexible and easy to modify. Figure 7 Illustrages how the same components can be used in differen OpenOffice.org applications[4].
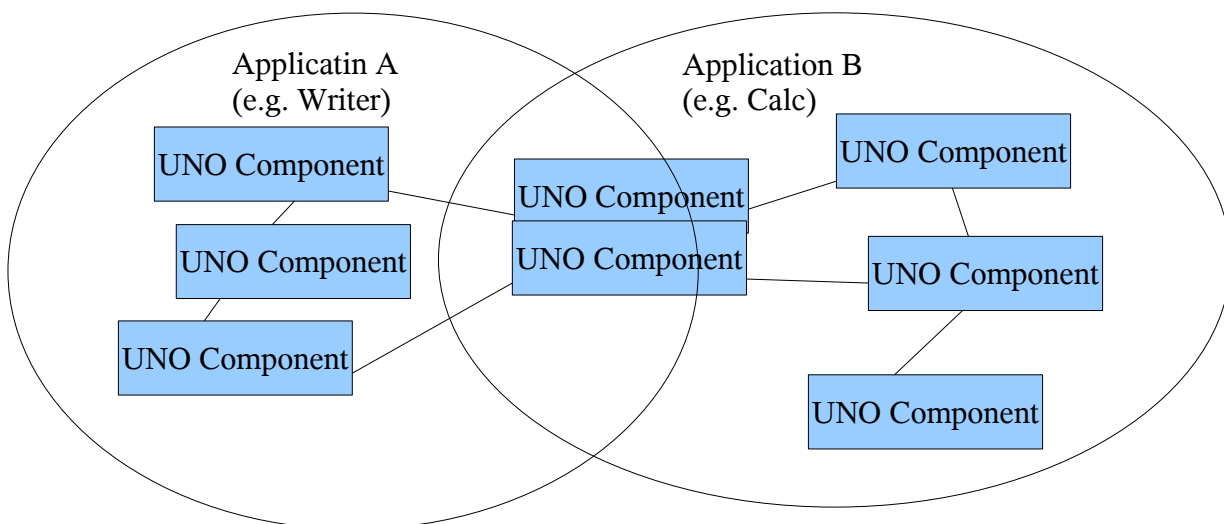
Figure 7: OO component based architecture.

---

[3] cp. [Burger05, p. 16].
[4] cp. [Burger05, p. 15].

2.4.4 OpenOffice.org API

The OpenOffice.org API is a language independent description of the OpenOffice.org functionality [devel05, p. 66]. Objects are described by their interfaces. By convention, all interfaces start with the letter 'X' (e.g. XDesktop). The API also uses special UNOIDL data types that are mapped to the specific programming languages to describe the exchanged data in an language independent way. The interfaces form the base for service specifications. Since one interface should only describe one aspect of an object (e.g. the object is "printable"), UNO uses multiple-inheritance interfaces in which more single-inheritance interfaces are grouped together to describe components [devel05, p. 69]. Furthermore, if an object shall be available as a general service in a global component context, an additional service description is needed that maps the service name to such a "grouped interface" [devel05, p. 69].
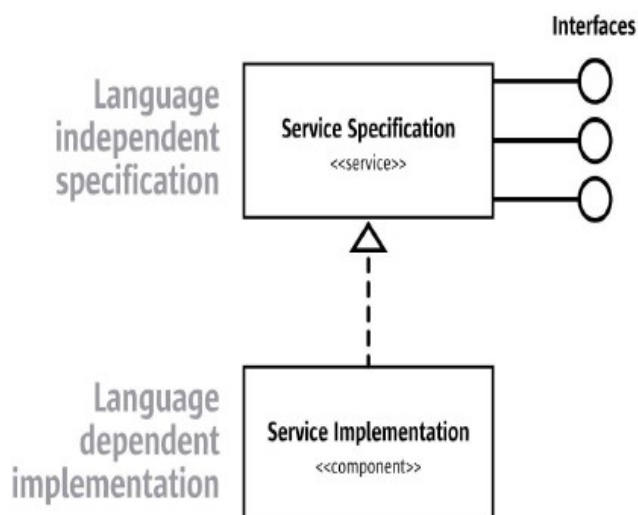
Figure 8: Interfaces, Service and Implementation [devel05, p. 70].

2.4.5 Service Managers

A service manager is the root object for connections to UNO and serves as an entry point for every UNO application. It is used to instantiate services by their service name, to enumerate all implementations of a certain service and to add or remove factories for a certain service at runtime. The service manager is passed to every UNO component during initialization [devel05, p. 88]. The main interface of the service manager is the *com.sun.star.lang.XMultiServiceFactory* interface. It offers three methods: *createInstance()* , *createInstanceWithArguments()* and *getAvailableServiceNames()* [devel05, p. 88].

2.4.6 Component Context

Often components need more functionality or information than a service manager can provide. Therefore the concept of the component context was created. A component context is basically a read only container for named values. One of these values is the service manager. The component context was designed to supersede the service manager as the
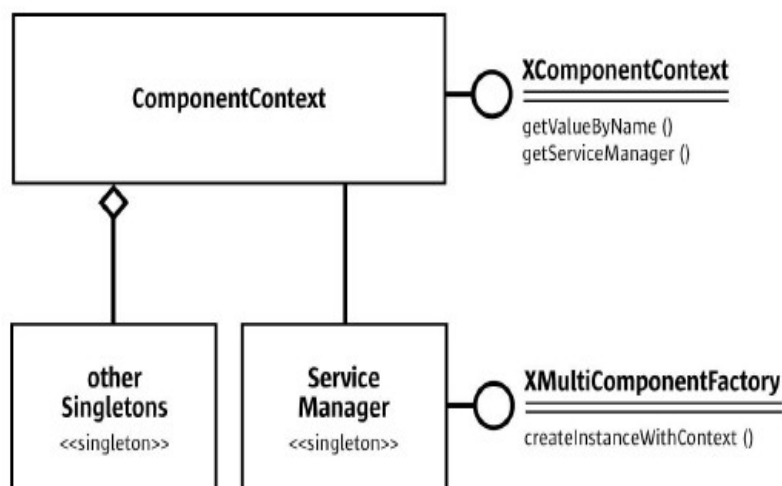
Figure 9: Component Context and Service Manager [devel05, p. 90].

central UNO object [devel05, p. 88f]. Figure 9 illustrates the connection between service manager and component context:

A component context only supports the *com.star.uno.XComponentContext* interface. This interface contains two Methods: *getValueByName* which returns a named value, and *getServiceManager* which returns the service manager [devel05, p. 90].

2.4.7 OpenOffice.org automation with BSF4Rexx

As seen in chapter 3, the Bean Scripting Framework can be utilized to handle Java classes with ooRexx. Via the Java language binding it is now also possible to use UNO components. To use the Java UNO class files the Java classpath must contain at least the files *jurt.jar, unoil.jar, ridl.jar* and *juh.jar* from the *<OfficePath>/program/classes* directory. The script *setEnvironment4OOo* that comes with the BSF4Rexx installation can help doing this. With *ScriptProviderforooRexx.jar,* which is also installed with BSF4Rexx, it is also possible to use ooRexx from within OpenOffice.org as a macro language. It only needs to be added in the menu 'Extras' – 'Package Manager' under 'My Packages'.

2.4.8 UNO.CLS

The ooRexx class UNO.CLS is a helper class that automates common steps when working with UNO. The line following line at the end of the ooRexx script makes this support available.

```
::requires UNO.CLS
```

The next code sample shows how UNO.CLS can be used to connect to OpenOffice.org:

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

The first line sums up a set of method calls that would be needed to connect to OpenOffice.org and receive the component context service. In the second line the *getServiceManager* Method is called and the component context returns a service manager object (implementing the *XMultiComponentFactory* interface). In the next step, the service manager can be used to get an instance of the desktop service manager, a service manager for documents that can be used to load documents and open OpenOffice.org applications.

2.5 Interaction of Components

The following figure shows how the whole system is linked.
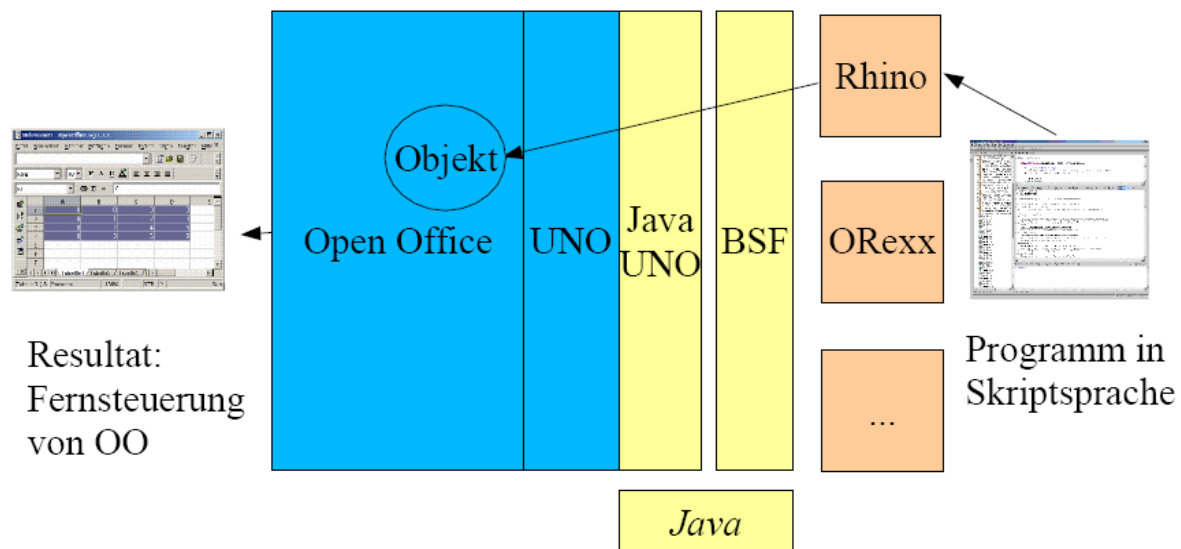
Figure 10: Concept of remote controlling OpenOffice.org with Rexx [Aug05].

The orange squares on the right side are standing for script languages, which are supportet by a BSF Framework (see Chapter 3.3 for details). BSF is the bridge between a script language and Java. Through BSF a script language can control Java.

The left, blue rectangle stands for Openoffice.org. It consists of several objects. The other blue rectangle symbolises the UNO interface of OpenOffice.org. It allows Java to access the objects of OpenOffice.org. So Java can control it.

This is the way how it works. Java controls Openoffice.org through UNO and Rexx controls Java using BSF. So Rexx can remote control OpenOffice.org.

2.6 How to Get a Running System

All the examples presented in this paper require ooRexx, Java and BSF4Rexx installed and working on the system. Additionally some of the examples requires the installation of external Java libraries.

In this chapter the installation and configuration of the necessary software is explained, including download links and tips for how more information and help can be found.

2.6.1 Java, OpenOffice.org and ooRexx

Java can be downloaded from http://java.sun.com/

The examples in this paper are tested with Java 1.4.2, but any newer version should work as well. It is recommended to use the latest released Java version.

OpenOffice.org can be downloaded from http://www.openoffice.org/.

OpenOffice.org is available in many languages, a list of all languages and links for downloading can be found at this address: http://projects.openoffice.org/native-lang.html. The examples in this paper is tested on version 2.0.2, but all subversions of version two should work too.

ooRexx can be downloaded from http://www.ooRexx.org/download.html.

The examples was developed on version 3.0.0. The newest stable version is 3.0.1 for Windows and version 3.0.0 for Unix. It is also a beta version of the 3.1 version[5] available.

---

[5]   ooRexx version 3.1 beta can be downloaded from this address:
http://sourceforge.net/project/showfiles.php?group_id=119701&package_id=130405&release_id=422462

2.6.2 BSF4Rexx

BSF4Rexx can be downloaded from the following address: http://wi.wu-wien.ac.at/rgf/Rexx/BSF4Rexx/current/.

The *readmeBSF4Rexx.txt* file also available from the link above, explain the installation of BSF4Rexx in detail.

The examples in this paper was tested on version 2.6 (2006-06-11). BSF4Rexx is active developed, and it is recommended to install the newest available version.

In case of Problems with installation or use of ooRexx or BSF4Rexx, the newsgroup "comp.lang.Rexx" might be of help.

2.6.3 External Java Libraries

Some of the examples presented in this paper use external Java libraries to gain additional functionality. E.g. the example C1-15 uses the external library JDOM in order to parse XML data.

External libraries need to be installed and made available to Java. All libraries available to Java, are also available to ooRexx with the use of BSF4Rexx.

The easiest way of doing this is to simply copy the external library into the "*lib/ext*" directory where Java is installed. On e.g. Suse Linux it is enough to copy the external library into this folder: "*/usr/lib/jre/lib/ext*".

This is the quickest and easiest method. However it might not be possible for normal users to do this, because normal users might not have the rights to write into the *"lib/ext"* directory of the Java installation.

An alternative method is to set the classpath to include the external library. The classpath is an environment variable that Java reads. The classpath variable points to where Java should look for the external libraries.

On a Linux system the content of the classpath variable can be inspected with the "*env*" command, and set with the *"export"* command

More about the two methods can be found at this link: http://mindprod.com/jgloss/classpath.html
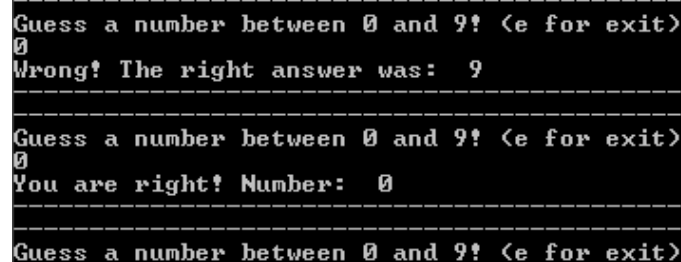
3 Examples

In this section the authors show how to use ooRexx, BSF4Rexx and OpenOffice.org by providing little examples, which introduce the reader to the key-features of these technologies.

3.1 C1 – Learning BSF4Rexx

The C1 examples cover the usage of ooRexx and BAF4Rexx. These examples show how to get access to and how to use classes of the standard Java API and external Java libraries.

3.1.1 Example C1-1 Java Randomizer Class

This example is a command line program. It asks the user to guess a number between 0 and 9. Afterwards it uses the Java Randomizer to create a number between 0 and 9 and compares it with the user input. Depending on the success of the user it writes an answer to the command line.

Figure 11: Output from example C1-1

```
/* import Java classes */
.bsf~bsf.import("java.util.Random","Random") /* import the Java Randomizer Class */

/* create objects */
r=.Random~new();


do while number<>"e"     /* Start a loop which ends if the user enters "e" */
        say "-------------------------------------------------" /* output to console */
        say "Guess a number between 0 and 9! (e for exit) " /* output to console */
        parse pull number                   /* input from console */
        x=r~nextInt(10)                      /* get a random number from the randomizer */
        if number=="e" then    /* control structure if user enters "e" */
                say "Good bye" /* output to console */
        else
                do
                        if number==x then   /* control structure if user entered the right number */
                                say "You are right! Number: " number /* output to console */
                        else
                                say "Wrong! The right answer was: " x /* output to console */
                end
        say "-------------------------------------------------" /* output to console */
end                         /* end of loop*/
/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 12: The code of example C1-1

3.1.1.1 Explanation

```
/* import Java classes */
.bsf~bsf.import("java.util.Random","Random") /* import the Java Randomizer Class */
```

The method "bsf.import" of the class "bsf" imports the "java.util.Random" Class in the program and assigns the name "Random" to it.

```
/* create objects */
r=.Random~new();
```

This line creates an instance of the "Random Class" with the name "r".

```
do while number<>"e"     /* Start a loop which ends if the user enters "e" */
```

This statement start a loop which ends if the variable "number" has the value "e". Later the user input will be assigned to this variable. So if the user enters "e" the loop will end.

```
        say "-------------------------------------------------" /* output to console */
        say "Guess a number between 0 and 9! (e for exit) " /* output to console */
```

Now write some output to the command line with the command "say".

```
parse pull number                   /* input from console */
```

This line reads the input of the user from the command line. The command "pull" reads the input and assigns it to the variable "number". By default Rexx converts the whole input to uppercase. Here the command "parse" is used to avoid this.

```
x=r~nextInt(10)                 /* get a random number from the randomizer */
```

This Statement executes the method "nextInt" of the Object "r" with the argument "10". This method creates a random number between 0 and the argument – 1. The result gets assigned to the variable "x".

```rexx
    if number=="e" then    /* control structure if user enters "e" */
            say "Good bye" /* output to console */
```

If the variable "number" has the value "e" then it writes "Good bye" to the command line.

```rexx
else
            do
                    if number==x then  /* control structure if user entered the right number */
                            say "You are right! Number: " number /* output to console */
                    else
                            say "Wrong! The right answer was: " x /* output to console */
            end
```

If the variable "number" has not the value "e", the program compares the variable "number" (user input) with the variable "x" (Random number from the Randomizer). If they are equal it writes "You are right! Number " and the number to the command line, if not it writes "Wrong! The right answer was " and the number.

This example is very simple and has no real difficulties. The only thing you have to be aware is, that Rexx converts all input from "pull" to uppercase, if you do not use "parse".

3.1.2 Example C1-2 – Regular Expressions

This example demonstrates how to use the Java "regexp" library to parse strings.

The package java.util.regexp provides classes for matching character sequences against patterns specified by regular expressions. The package contains two classes: the class *Pattern* for representing regular expressions and the class *Matcher* for matching character sequences against patterns [regexp06].

```
/* import Java classes */
.bsf~bsf.import("java.util.regex.Pattern", "Pattern")
.bsf~bsf.import("java.util.regex.Matcher", "Matcher")

str = "Did you know: You can use message boxes with the class bsf.dialog. Really!"
say str

pattern = .Pattern~compile(": .*[.]")
matcher = pattern~matcher(str)


if matcher~find() <> 0 then do
        say  matcher~group()
end
else say "No match found."

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 13: The code of example C1-2.

3.1.2.1 Explanation

First, the necessary classes *Pattern* and *Matcher* from the Java package java.util.regexp are imported. Then a string '*str*' with sample data is defined.

The following two lines create two objects: first an instance of the class *Pattern* is created. This is done by a call to the static class method *compile*, with a regular expression pattern as parameter. Then a call to the pattern objects method *matcher* returns an object of the type *Matcher* which holds the string '*str*' as input data:

```
pattern = .Pattern~compile(": .*[.]")
matcher = pattern~matcher(str)
```

The matcher objects method 'find' attempts to find a subsequence in the matchers input data:

```
if matcher~find() <> 0 then do
        say  matcher~group()
end
```

The 'find' method returns a boolean value indicating if a matching substring was found. If so, the method 'group' returns the matching subsequence. If the find method would be

called again, it would try to find the next matching substring in the input data and group would return the next matching part.

This example has the following output:

```
> Rexx regexp_simple.rex

Did you know: You can use message boxes with the class bsf.dialog.
Really!

: You can use message boxes with the class bsf.dialog.
```

Figure 14: Output of example C1-2.

The first line is the string with the input data, the second output line is the substring that matches the specified pattern returned by the *group* method.

Another example how to use this package can be found in example C3-2.

3.1.3 Example C1-3 Math

The following code demonstrates how to call static Java methods using BSF4Rexx. This is done using some of the static methods in the java.lang.Math library. A static method is a method that is directly available, and you do not need to create an object in order to use the method.
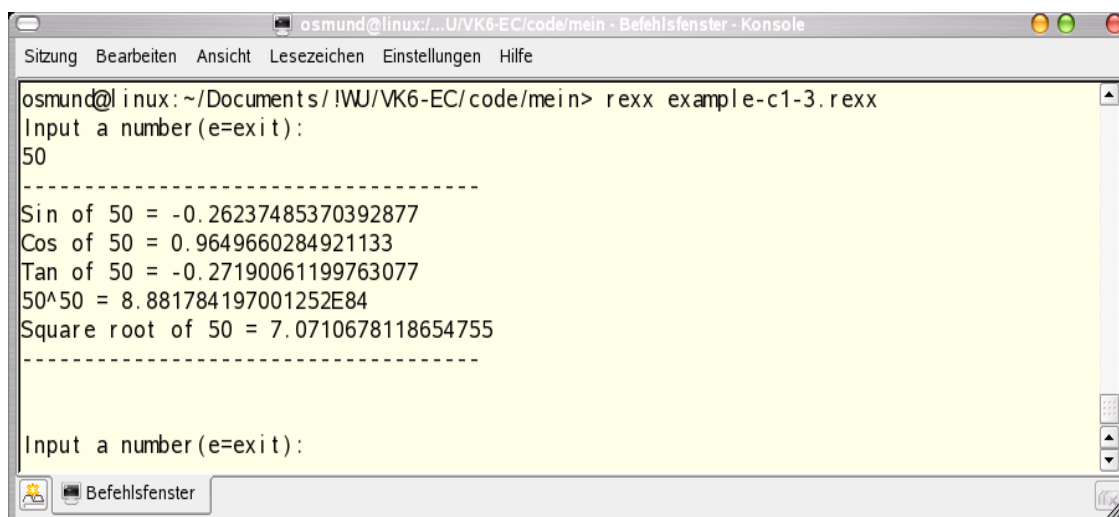
In the following example the user is asked to input a number. If the user types an "e" or "E", the script is terminated. If the user inserts a number, the sinus, cosines, tangent, square and the number raised to the power of itself is calculated and printed on the screen.

```rexx
/* import classes */
.bsf~bsf.import("java.lang.Math","Math")

do forever
    say "Input a number(e=exit):"
    pull number
    if number="E" then leave
    say "-----------------------------------"
    /* This is all static methods, hence we do not create an object - just call the method */
    say "Sin of "number" = ".Math~sin(number)
    say "Cos of "number" = ".Math~cos(number)
    say "Tan of "number" = ".Math~tan(number)
    say number"^"number" = ".Math~pow(number,number)
    say "Square root of "number" = ".Math~sqrt(number)
    say "-----------------------------------"
    say
    say
end

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 15: The code for example C1-3

Figure 16: Output from example C1-3

3.1.3.1 Explanation

At the last line of the script, the BSF.CLS file is loaded. This is essential in order to use BSF4Rexx. An alternatively method is to load UNO.CLS, that is specially made for the purpose of supporting ooRexx and BSF4Rexx development in relation to OpenOffice.org. UNO.CLS automatically loads BSF.CLS. It is sufficient for scripts that do not work with OpenOffice.org only to load BSF.CLS

UNO.CLS and the use of ooRexx in relation to OpenOffice.org is covered in detail later in this paper.

On the the second line of the script the java.lang.Math library is loaded and made available. The name *Math* can be used later in the script in order to refer to this library.

Inside the internal loop (do forever ... end) the user is prompted for a number, and the number is passed to some static methods in the imported Java library.

```
say "Sin of "number" = ".Math~sin(number)
say "Cos of "number" = ".Math~cos(number)
say "Tan of "number" = ".Math~tan(number)
say number"^"number" = ".Math~pow(number,number)
say "Square root of "number" = ".Math~sqrt(number)
```
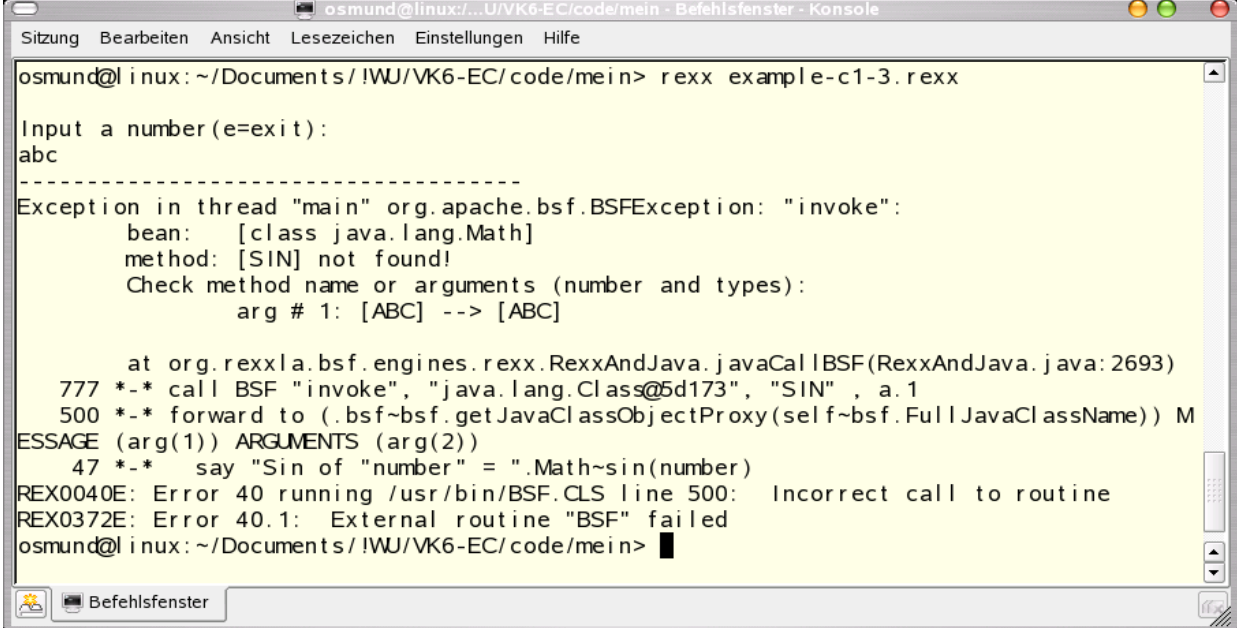
The code used in this example is just a simple extract of the possible mathematics methods available to ooRexx with the help of BSF4Rexx. A complete list of all the functions in the java.lang.Math library can be found in the API documentation.[6]

```
if number="E" then leave
```

Inside the loop, a check is done if the input value is the character "E" or "e". If that is the case the script will exit. It is only necessary to check for an capital "E", because the "pull" command used to retrieve the input value converts the input to capital letters.

If the user gives the script any input different from a number or an "e", the script will fail. The reason for this is that all the java.lang.Math methods used in this example requires either an integer or a double value.

---

[6] The API documentation for java.lang.Math can be found here:
http://java.sun.com/j2se/1.5.0/docs/api/index.html

```
osmund@linux:~/Documents/!WU/VK6-EC/code/mein> rexx example-c1-3.rexx

Input a number(e=exit):
abc
-------------------------------------
Exception in thread "main" org.apache.bsf.BSFException: "invoke":
        bean:    [class java.lang.Math]
        method: [SIN] not found!
        Check method name or arguments (number and types):
                arg # 1: [ABC] --> [ABC]

        at org.rexxla.bsf.engines.rexx.RexxAndJava.javaCallBSF(RexxAndJava.java:2693)
   777 *-* call BSF "invoke", "java.lang.Class@5d173", "SIN" , a.1
   500 *-* forward to (.bsf~bsf.getJavaClassObjectProxy(self~bsf.FullJavaClassName)) M
ESSAGE (arg(1)) ARGUMENTS (arg(2))
    47 *-*   say "Sin of "number" = ".Math~sin(number)
REX0040E: Error 40 running /usr/bin/BSF.CLS line 500:  Incorrect call to routine
REX0372E: Error 40.1:  External routine "BSF" failed
osmund@linux:~/Documents/!WU/VK6-EC/code/mein> █
```

Figure 17: Error message from example C1-3

If the user enters e.g. "abc" instead of a number, BSF4Rexx will try to find a method called e.g. "sin" with the input value of type string within the java.lang.Math library. This method do not exist, and BSF4Rexx will return the message bellow.

The error message contains the text **"method: [SIN] not found !"**, and the developer might think that he or she used the wrong method name, or that something is wrong with the import of the library. But a second inspection of the error message reveals the following text that gives a hint about the real problem: **"Check method name or arguments (number and types)"**.

It is possible to catch all errors in ooRexx with the "signal on" commands, as demonstrated in example-C1-21. If the user enters a valid value, the following feedback will be given.

3.1.4Example C1-4 Java awt and swing Classes

This example demonstrates how to create a GUI (Graphical User Interface). It contains 2 Radiobuttons and a label which shows different icons containing GIFs (Graphic Interchange Format) depending which radiobutton is activated.
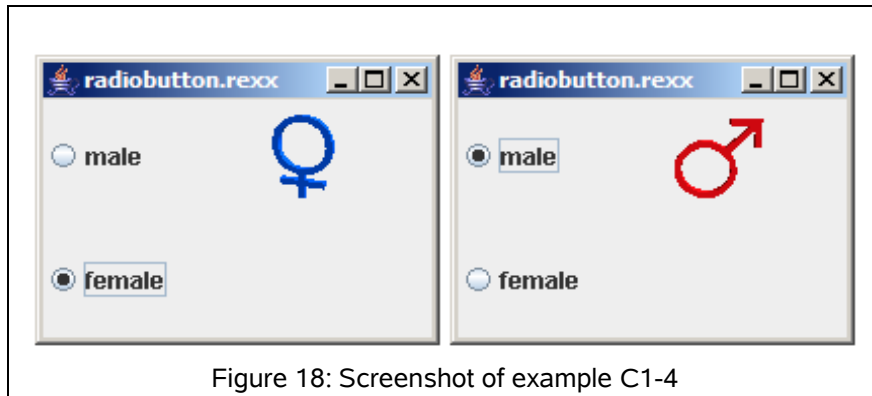
Figure 18: Screenshot of example C1-4

```
/* import Java classes */
.bsf~bsf.import("javax.swing.JFrame","JFrame") /* import the Java Swing JFrame class */
.bsf~bsf.import("javax.swing.JLabel","JLabel")  /* import the Java Swing JLabel class */
.bsf~bsf.import("javax.swing.JRadioButton","JRadio") /* import the Java Swing JRadiobutton class */
.bsf~bsf.import("java.awt.GridLayout","GridLayout") /* import the Java Awt Gridlayout class */
.bsf~bsf.import("javax.swing.ImageIcon","Icon")  /* import the Java Swing Imageicon class */

/* create objects */
label=.JLabel~new("");
frame = .JFrame~new("radiobutton.Rexx");
male = .JRadio~new("male");
female = .JRadio~new("female");
mIcon = .Icon~new("male.gif");
fIcon = .Icon~new("female.gif");

frame~getContentPane()~setLayout(.GridLayout~new(2,2,5,5));
/* set the the frame to use the GridLayout */

frame~getContentPane()~~add(male)~~add(label)~~add(female);
/* Add label, icon and radiobuttons to frame */

/* Add eventhandling */
frame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"));
frame~bsf.addEventListener('window', 'windowClosing', 'call BSF "exit"')
male~bsf.addEventListener('action', '', 'call setmale')

female~bsf.addEventListener('action', '', 'call setfemale')

frame~setLocation(500,300); /* set the location of the frame on the screen */
frame~~pack()~~setSize(200,150)~~setVisible(.true); /* Set the size of the frame and show it */

/* a never ending loop which waits for messages of the eventhandler */
do forever
   a = bsf("pollEventText")     /* wait for messages from the eventhandler     */
   interpret a    /* execute the message from the eventhandler as a Rexx program    */
end
/* this procedure is called from the eventhandler for the "male" button */
setmale:
    label~setIcon(mIcon);
    female~setSelected(".false");
return

/* this procedure is called from the eventhandler for the "female" button */
setfemale:
    label~setIcon(fIcon);
    male~setSelected(".false");
return
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 19: The code of example C1-4

3.1.4.1 Explanation

```
/* import Java classes */
.bsf~bsf.import("javax.swing.JFrame","JFrame") /* import the Java Swing JFrame class */
.bsf~bsf.import("javax.swing.JLabel","JLabel")  /* import the Java Swing JLabel class */
.bsf~bsf.import("javax.swing.JRadioButton","JRadio") /* import the Java Swing JRadiobutton class */
.bsf~bsf.import("java.awt.GridLayout","GridLayout") /* import the Java Awt Gridlayout class */
.bsf~bsf.import("javax.swing.ImageIcon","Icon")  /* import the Java Swing Imageicon class */
```

The method "bsf.import" of the class "bsf" imports the following class in the program and assigns a name to it.

- javax.swing.JFrame

- javax.swing.JLabel

- javax.swing.JRadioButton

- java.awt.GridLayout

- javax.swing.ImageIcon

```
/* create objects */
label=.JLabel~new("");
frame = .JFrame~new("radiobutton.Rexx");
male = .JRadio~new("male");
```

```
female = .JRadio~new("female");
mIcon = .Icon~new("male.gif");
fIcon = .Icon~new("female.gif");
```

These lines creates an instances of the imported Classes.

```
frame~getContentPane()~setLayout(.GridLayout~new(2,2,5,5));
```

This statement forces the frame to use "Grid Layout".

```
frame~getContentPane()~~add(male)~~add(label)~~add(female);
```

Add the radiobuttons and the label to the frame.

```
frame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"));
frame~bsf.addEventListener('window', 'windowClosing', 'call BSF "exit"')
```

Add eventhandler which exits the program if the user clicks on the close icon of the window.

```
male~bsf.addEventListener('action', '', 'call setmale')
female~bsf.addEventListener('action', '', 'call setfemale')
```

Add eventlistener which starts procedures, which are defined a few lines lower. These procedures are named "setmale" and "setfemale". They are called with the command "call".

```
frame~setLocation(500,300); /* set the location of the frame on the screen */
```

Set the position, where the window should appear on the screen.

```
frame~~pack()~~setSize(200,150)~~setVisible(.true); /* Set the size of the frame and show it */
```

Set the size of the window and make it visible on the screen.

```
do forever
   a = bsf("pollEventText")      /* wait for messages from the eventhandler     */
   interpret a      /* execute the message from the eventhandler as a Rexx program    */
end
```

Start a never ending loop. Inside of the loop the messages of the eventhandler are executed as Rexx program lines.

```
/* this procedure is called from the eventhandler for the "male" button */
setmale:
    label~setIcon(mIcon);
    female~setSelected(".false");
return

/* this procedure is called from the eventhandler for the "female" button */
setfemale:
    label~setIcon(fIcon);
    male~setSelected(".false");
return
```

Procedures are defined just with their name and a ":". In the last line there must be a "return" statement. The first line inside the procedures assigns the icon to the frame, the second line set the other radiobutton unselected.

This example is simple and has no real difficulties, if the programmer is familiar with the usage of Java awt and Swing.

3.1.5 Example C1-5 Message Boxes

This example displays various types of Input-, Dialog and Message Boxes.

Message Boxes are a simple way to make scripts and macros more user-friendly. This example demonstrates two simple and easy ways to use them with ooRexx.

```
/* import JoptionPane from Swing */
.bsf~bsf.import("javax.swing.JOptionPane", "JOptionPane")
/* JOptionPane Message Boxes:    */
.JOptionPane~showMessageDialog(.nil, "Message Box demo begins!", "Info", .JOptionPane~WARNING_MESSAGE)
.JOptionPane~showMessageDialog(.nil, "BSF4Rexx is great!")
/* YES_NO Dialog */
answer = ""
do while answer <> .JOptionPane~YES_OPTION
        answer = .JOptionPane~showConfirmDialog(.nil, "do you like Rexx?", "choose one", -
.JOptionPane~YES_NO_OPTION)
        if (answer == .JOptionPane~YES_OPTION) then say yes
        if (answer == .JOptionPane~NO_OPTION) then say no
end
  /* Modal dialog with OK cancel and a text field */
    text = .JOptionPane~showInputDialog("Tell me something:", "type here..")
    if (text == .nil) then say cancel
    else say text;
/* Other way to display MessageBoxes: using .bsf.dialog */
answer = .bsf.dialog~inputBox("Tell me more","BSF Input Box","question" )
say answer

answer = .bsf.dialog~dialogBox("Options for you:",, "BSF DialogBox", "YesNoCancel" )
say answer

answer = .bsf.dialog~messageBox("I'm a BSF MessageBox", "Important message:","plain");
say answer
/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 20: Output of example C1-5.

3.1.5.1 Explanation

The Java class *JOptionPane* offers various static class methods to display different types of Input-, Dialog- and Message Boxes without having to create extra objects. In the first line of the code, this class is imported to be used in the Rexx script:

```
.bsf~bsf.import("javax.swing.JOptionPane", "JOptionPane")
```

In the following lines, two message boxes are created via calls to the static *showMessageDialog* function with different parameters. Figures 21 and 22 show the results:
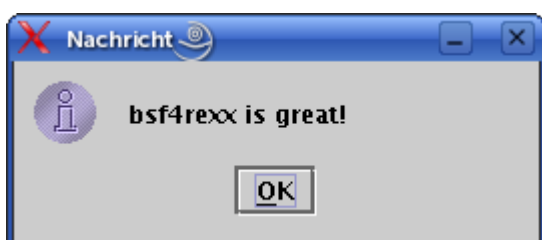
Figure 21: JOptionPane - MessageBox 2
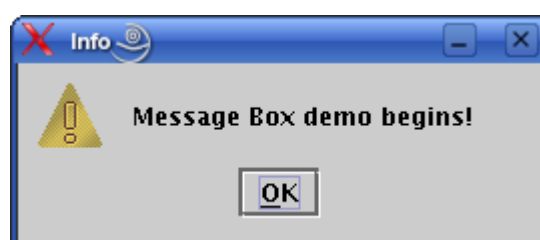
Figure 22: JOptionPane - MessageBox 1

The following loop shows a dialog box again and again until the user presses the "Yes" Button. Figure 23 shows the output of this code part. The static method *showConfirmDialog* of *JOptionPane* is therefore used. The method returns an integer value which is compared to *JOptionPane* (constant) class variables like "YES_OPTION" or "NO_OPTION". This example also demonstrates how to access class constants. The *.nil* object as first parameter represents the parent frame and equals to the *null* value in Java:

```
        answer = .JOptionPane~showConfirmDialog(.nil, "do you like Rexx?", "choose one", -
.JOptionPane~YES_NO_OPTION)
        if (answer == .JOptionPane~YES_OPTION) then say yes
```


Figure 23: JOptionPane - DialogBox

Another practical feature of *JOptionPane* are input dialogs. They can be used to receive some text input from the user in a convenient way:

```
text = .JOptionPane~showInputDialog("Tell me something:", "type here..")
```
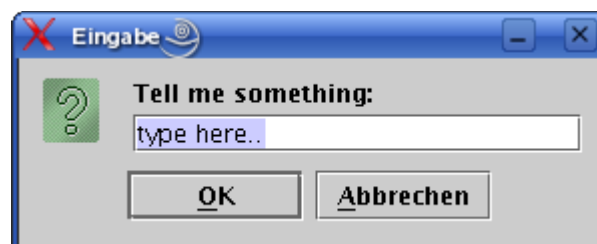

Figure 24: JOptionPane - InputBox

This method returns a string Java value. If the user entered no text, the result will be *.nil*.

Another, even more simple way to produce the same results as before is by using the class BSF.DIALOG, provided by BSF.CLS. This class provides the methods *messageBox, dialogBox* and *inputBox*. Figures 25 till 27 show the results of these methods.
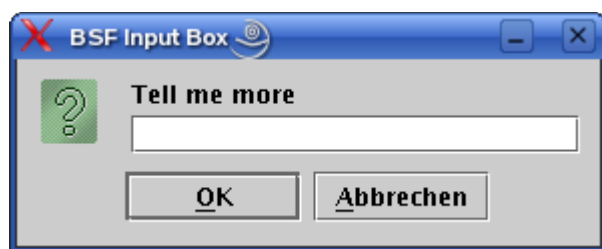
Figure 26: BSF.DIALOG InputBox
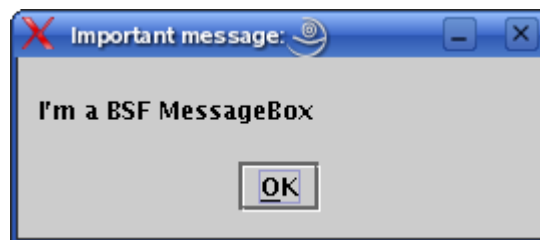
Figure 25: BSF.DIALOG - MessageBox
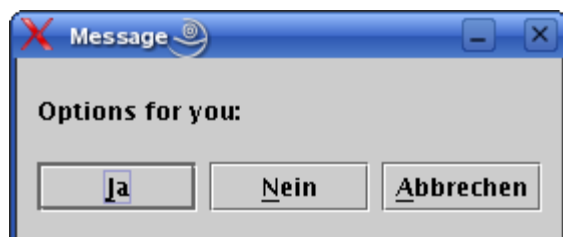
Figure 27: BSF.DIALOG - DialogBox

Internally, the BSF.DIALOG class also uses the Java class *JOptionPane,* but it provides the functionality in a simpler way.

3.1.6 Example C1-6 Simple Swing

The purpose of this example is to demonstrate how to use the swing library to create nice looking Graphical User Interfaces. (GUI)

Swing is a built-in library in Java and can be used to build large an complex GUIs.
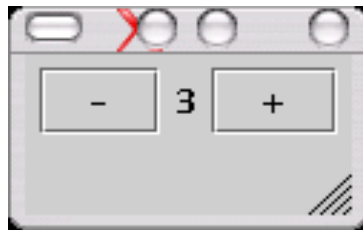
Figure 28: Screenshot of
example C1-6

This example display a window with two buttons and label. In the label a number is displayed. If the user pushes the "+" button, the event is caught from the script, and dispatched to a method that increases the number displayed in the label. If the user pushes the "-" button, the number in the label will decrease.

Additionally, every time the user selects one of the buttons, a text will be written to the console (standard out).

```rexx
/* import classes */
.bsf~bsf.import("javax.swing.JFrame","JFrame")
.bsf~bsf.import("javax.swing.JButton","JButton")
.bsf~bsf.import("javax.swing.JLabel","JLabel")
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout")

/* create objects */
buttonPlus=.JButton~new("+");
buttonMinus=.JButton~new("-");
number=.JLabel~new("0");
frame = .JFrame~new("My Counter Frame !");

/* set the the frame to use the FlowLayout */
frame~getContentPane()~setLayout(.FlowLayout~new());

/* Add label and buttons to frame */
frame~getContentPane()~~add(buttonMinus)~~add(number)~~add(buttonPlus);


/* Add eventhandling */
frame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"));
frame~bsf.addEventListener('window', 'windowClosing', 'call BSF "exit"')
buttonPlus~bsf.addEventListener('action', '', 'call increase')
buttonMinus~bsf.addEventListener('action', '', 'call decrease')

/* And finally show the frame .. */
frame~~pack()~~setVisible(1);


/* do a loop and wait for the event handling to return with some text */
do forever
   a = bsf("pollEventText")     /* wait for an eventText to be sent     */
   say "I got this text from the event handler :" a
   interpret a                  /* execute as a Rexx program     */
end


/* this procedure is called from the event handler for the plus button */
increase:
    number~setText(number~getText()+1);
    frame~pack();
return

/* this procedure is called from the eventhandler for the minus button */
decrease:
    number~setText(number~getText()-1);
    frame~pack();
return

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 29: The code of example C1-6

3.1.6.1 Explanation

```
/* import classes */
.bsf~bsf.import("javax.swing.JFrame","JFrame")
.bsf~bsf.import("javax.swing.JButton","JButton")
.bsf~bsf.import("javax.swing.JLabel","JLabel")
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout")
```

In the first lines of the script BSF4Rexx is instructed to import some libraries. In Java it is possible to import all classes in a library in one command. This is not possible with BSF4Rexx, hence all the classes have to be imported one by one.

The *JFrame* class is the window displayed to the user, and the *JButton* and *JLabel* are the buttons and the label added to, and displayed on the window.

All the imported swing classes represent a visible object for the user, the imported *FlowLayout* on the other hand, is not a visible object, but a description of how the objects are organized on the screen. Every time the *JFrame* is updated it asks the associated layout manager for information about how to display the elements. *FlowLayout* is the layout manager used in this example, and it specifies that all the objects added to the *JFrame* should be ordered from left to right. But other layout managers exist as well, and they could be used in order to achive a different look of the GUI.[Ull05]

Under two other often used layout mangers are presented.

- The BorderLayout divides the available area in 5 sections: North, South, East, West and Middle. A visual object can be assigned to each of the 5 areas.

- GridLayout organizes all the components into equal size, and insert them into a grid. The programmer can define the dimension of the grid using the constructor of the GridLayout. The dimension is defined by specifying the numbers of rows and columns to use.

More information about layout managers and how to use them can be found on the page "A Visual Guide to Layout Managers" on the Sun website.[7]

---

[7]   http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html

The next part of the code create objects of the imported classes. This is done by calling the method new from the class. If an argument is given to the method new, it will be passed on to the constructor of the class.

```
buttonPlus=.JButton~new("+");
buttonMinus=.JButton~new("-");
number=.JLabel~new("0");
frame = .JFrame~new("My Counter Frame !");
```

Two objects, *buttonPlus* and *buttonMinus*, are created from the class *JButton*. The argument passed on to the constructor is the caption of the button. The label and the frame are created in the same way as the buttons. Be aware that even though all the visual objects are created, they are not yet associated with the frame, and will not be displayed on the screen before this is done.

```
frame~getContentPane()~setLayout(.FlowLayout~new());
```

Next in the example, the ContentPane of *frame*, need to be instructed to use the correct layout manager. This is done by first getting the *ContentPane* object, and then call the method *setLayout* on this object.

Using BSF4Rexx it is possible to connect a method to the result of another method using the ~ ("Twiddle") sign. An alternative to the code above could be to store the *ContentPane* object in a variable, and then call the *setLayout* method on this object. The code block below shows this alternative method:

```
/* set the the frame to use the FlowLayout – alternative method*/
CP = frame~getContentPane()
CP~setLayout(.FlowLayout~new());
```

After the *ContentPane* has set the layout manager, the visual objects are added. As in case of the layout manager, visual objects are also applied to the *frame's ContentPane*.

```
/* Add label and buttons to frame */
frame~getContentPane()~~add(buttonMinus)~~add(number)~~add(buttonPlus);
```

If a method is connected with the signs ~~, this is a short way of repeating the whole statement. Hence the code block above is equal the code block bellow.

```
/* Add label and buttons to frame – alternative method */
frame~getContentPane()~add(buttonMinus)
frame~getContentPane()~add(number)
frame~getContentPane()~add(buttonPlus)
```

After the visual objects are added to the frame, the event handling needs to be specified.

```
/* Add eventhandling */
frame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"));
frame~bsf.addEventListener('window', 'windowClosing', 'call BSF "exit"')
buttonPlus~bsf.addEventListener('action', '', 'call increase')
buttonMinus~bsf.addEventListener('action', '', 'call decrease')
```

First it is specified that the application has to be terminated if the user closes the window. Then an eventlistener for the *frame* is set.

The two lines at the end specify the eventlistener for the buttons. The third argument is the code that has to be executed when the button is pressed. In this case the ooRexx routine *increase* is executed if the *buttonPlus* is pressed.

```
/* do a loop and wait for the event handling to return with some text */
do forever
   a = bsf("pollEventText")      /* wait for an eventText to be sent    */
   say "I got this text from the event handler :" a
   interpret a                   /* execute as a Rexx program    */
end
```

After the GUI is created the script goes into an eternal loop, only aborted if the user exit the *frame*. The command *bsf("pollEventText")* waits for the event handler to parse an event.

The event the event handler will return is the third argument of the *bsf.addEventListener* statement. If an event occurs it is written into the variable *a* and printed on the console using the ooRexx say command.

The ooRexx command *interpret* executes the content of variable *a* as ooRexx code.

If the user press the plus button the ooRexx routine *increase* will be called. This routine gets the number in the label, increase it by one, and write it back into the label.

3.1.7 Example C1-7 Java Midi Classes

This example plays a short song using the Java Midi Classes. This example is based on a Java example. [Kru04]

```
/* import Java classes */
.bsf~bsf.import("javax.sound.midi.MidiSystem","MidiSystem") /* import Java MidiSystem class */
.bsf~bsf.import("javax.sound.midi.ShortMessage","Message")
                                                /* import Java Midi Shortmessage class */
.bsf~bsf.import("java.awt.Robot","Robot") /* import Java awt Robot class */

/* compose song by inserting notes in an array*/
song.1.1=60;  song.1.2=1;  song.1.3=1
song.2.1=62;  song.2.2=1;  song.2.3=1
song.3.1=64;  song.3.2=1;  song.3.3=1
song.4.1=65;  song.4.2=1;  song.4.3=1
song.5.1=67;  song.5.2=2;  song.5.3=2
song.6.1=69;  song.6.2=1;  song.6.3=4
song.7.1=67;  song.7.2=4;  song.7.3=1
song.8.1=69;  song.8.2=1;  song.8.3=4
song.9.1=67;  song.9.2=4;  song.9.3=1
song.10.1=65; song.10.2=1; song.10.3=4
song.11.1=64; song.11.2=2; song.11.3=2
song.12.1=62; song.12.2=1; song.12.3=4
song.13.1=60; song.13.2=4; song.13.3=1

/* get the Java Midi Synthesizer */
Synthesizer=.MidiSystem~getSynthesizer();
Synthesizer~open()
/* get the Receiver of the Java Midi Synthesizer */
Receiver=Synthesizer~getReceiver()

/* create new Message object */
msg = .Message~new()
rbt = .Robot~new()
DO note=1 to 13
        DO count=1 to song.note.3
                msg~setMessage(144, 0, song.note.1, 64)
                                                /* Define Message to start playing a tone*/
                Receiver~send(msg, -1) /* Send Message to the Receiver */
                rbt~delay(song.note.2*400) /* wait the time a tone should be played */
                msg~setMessage(128, 0, song.note.1, 64)
                                        /* Define Message to stop playing the tone*/
                Receiver~send(msg, -1) /* Send Message to the Receiver */
        END
END
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 30: The code of example C1-7

3.1.7.1 Explanation

```
/* compose song by inserting notes in an array*/
song.1.1=60;  song.1.2=1;  song.1.3=1
song.2.1=62;  song.2.2=1;  song.2.3=1
song.3.1=64;  song.3.2=1;  song.3.3=1
song.4.1=65;  song.4.2=1;  song.4.3=1
song.5.1=67;  song.5.2=2;  song.5.3=2
song.6.1=69;  song.6.2=1;  song.6.3=4
song.7.1=67;  song.7.2=4;  song.7.3=1
song.8.1=69;  song.8.2=1;  song.8.3=4
song.9.1=67;  song.9.2=4;  song.9.3=1
song.10.1=65; song.10.2=1; song.10.3=4
song.11.1=64; song.11.2=2; song.11.3=2
song.12.1=62; song.12.2=1; song.12.3=4
song.13.1=60; song.13.2=4; song.13.3=1
```

In this section an 2-dimensional array is defined. The structure is as follows:

"name of the array"."position in the first dimension"."position in the second dimension"="value"

The first dimension indicates the sequence the notes are played.In the second dimension the first element defines the frequence of the note, second one shows how long a note has to be played and the third one defines how often a note should be played at this moment.

```
Synthesizer=.MidiSystem~getSynthesizer();
Synthesizer~open()
```

Getting the Java Synthesizer by using the static method "getSynthesizer" of the Java class "MidiSystem"

```
Receiver=Synthesizer~getReceiver()
```

Get the Receiver of the Synthesizer

```
msg = .Message~new()
rbt = .Robot~new()
```

Create instances of these classes.

```
msg~setMessage(144, 0, song.note.1, 64) /* Define Message to start playing a tone*/
Receiver~send(msg, -1) /* Send Message to the Receiver */
```

Assign a message to the message object. The argument "144" causes starting to play the "song.note.1" note. Afterwards the message is send to the Receiver.

```
rbt~delay(song.note.2*400) /* wait the time a tone should be played */
```

Wait the time a note should be played.

```
msg~setMessage(128, 0, song.note.1, 64)    /* Define Message to stop playing the tone*/
Receiver~send(msg, -1) /* Send Message to the Receiver */
```

Send another message to the Receiver, with sending the argument "128" the playback of the note gets stopped.

This example is simple and has no real difficulties, if the programmer is common with the usage of Java Midi Classes. There is a problem with the newest Java version 1.5. No error messages are shown, but no sound is played. This example was tested with Java 1.4.2 and with this version it works fine.

3.1.8 Example C1-8 Reflection

This example shows how to get information about objects by using the reflection API.

The reflection API allows to [reflect06]:

- Determine the class of an object.

- Get information about a class's modifiers, fields, methods, constructors, and superclasses.

- Find out what constants and method declarations belong to an interface.

- Create an instance of a class whose name is not known until runtime.

- Get and set the value of an object's field, even if the field name is unknown to your program until runtime.

- Invoke a method on an object, even if the method is not known until runtime.

- Create a new array, whose size and component type are not known until runtime, and then modify the array's components.

This features might be interesting for debugging scripts that use BSF4Rexx or writing more dynamic scripts and macros.

```
/* import standard awt class TextField */
.bsf~bsf.import("java.util.regex.Pattern", "Pattern")

/* create a test object */
object = .Pattern~compile(":.*")
class = object~getClass()
superclass = class~getSuperclass()

/* get class and superclass name */
say "Class name:" class~getName()
say "Superclass: " superclass~getName

/* list the classes methods */
say "Methods:"
  methods = class~getMethods()
  do method over methods
       say "------------------------"
       say "Method:" method~getName
       say "Return type:" method~getReturnType~getName
       ptypes = method~getParameterTypes()
       do type over ptypes
            say "Parameter:" type~getName
       end
end
/* list the classes fields */
say "Fields:"
fields = class~getFields
do field over fields
       say field~toString
end
/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 31: The code of example C1-8.

3.1.8.1 Explanation

This example uses an object of the class *Pattern* (see also example C1-2) as a test object. In the first part of the code, the necessary Java classes are imported and the object is created.

```
> Rexx Reflection.rex
Class name: java.util.regex.Pattern
Superclass:  java.lang.Object
Methods:
------------------------
Method: compile
Return type: java.util.regex.Pattern
Parameter: java.lang.String
------------------------
.......
Fields:
public static final int java.util.regex.Pattern.UNIX_LINES
public static final int java.util.regex.Pattern.CASE_INSENSITIVE
......
```

Figure 32: Output of example C1-8

The method *getClass* returns an object of the class *java.lang.Class*:

```
class = object~getClass()
```

Via this object, the relevant information about the test objects class can be accessed. The Method *getSuperclass* returns a *Class* object representing the test objects parent class. *getName* returns the class name for each *Class* object (i.e. the objects class, the parent class and so on).

```
methods = class~getMethods()
```

This line returns an array of objects with class *java.lang.reflect.Method*. Via these objects the relevant information about the classes methods can be retrieved. The Method *getParameterTypes* returns an array of *java.lang.Class* objects that describe the methods parararameters. Also the return type of *getReturnType* is a *Class* object.

The class objects method *getFields* returns an array of *java.lang.reflect.Field* objects. These objects can be used to get information about the classes fields, i.e. *getName* returns the name of a field, but they also have get- and set-methods for manipulating field values.

3.1.9 Example C1-9 Hash

The following example reads a file and create a SHA-1 hash value for it.

```
/* import classes */
.bsf~bsf.import("java.security.MessageDigest","MessageDigest")
.bsf~bsf.import("java.io.File","File")
.bsf~bsf.import("java.io.FileInputStream","FileInputStream")
.bsf~bsf.import("java.lang.Byte","Byte")

/* create the messageD object using the static method getInstance */
messageD = .MessageDigest~getInstance("SHA-1")

/* create a byte array with the size of 8192 bytes */
md =  bsf.createArray("byte.class",8192)

/* create inputstream from file */
inStream = .FileInputStream~new(.File~new("example-C1-9.Rexx"))

/* this loop read 8192 bytes from file and updates hash, loop until eof */
DO FOREVER
    n=inStream~read(md) /* stream returns -1 if EOF */
    if n=-1 then leave  /* exit loop if EOF */
    messageD~update(md,0,n) /* update the hash value */
END

hash = messageD~digest()
size = hash~items
hashPrint = ""

i=1
/* this loop convert the resulting hash from byte to hex */
DO until i>size
    b=.Byte~new(hash[i])~toString()
    hashPrint = hashPrint d2x( b ,2)
    i=i+1
END

/* print the final result */
say "SHA-1(160 bit) hash of this file is :"
say hashPrint

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 33: The code for example C1-9

3.1.9.1 Explanation

In the first part of the script the libraries needed libraries are imported. All the libraries are present in the standard Java API.

```
/* create the messageD object using the static method getInstance */
messageD = .MessageDigest~getInstance("SHA-1")
```

Next a message digest object is creaded using the static method *getInstance*. Using this method several other algorithms for hash creation can also be used. In Java version 1.5 the following the following algorithms are supported: [Sun04]

- MD2

- MD5

- SHA-1

- SHA-256

- SHA-348

- SHA-512

Even though MD5 might be the most famous algorithm, SHA-x algorithms are consider safer in order to avoid problems with collisions. MD2 are not longer considered secure, and should not be used, unless for compatibility reasons. [Wik06-1] [Wik06-2]

```
/* create a byte array with the size of 8192 bytes */
md =  bsf.createArray("byte.class",8192)
```

Next a byte array of 8192 bytes is created. The byte array could be as long as the data to hash, but because of performance reasons it is recommended to divide the data into smaller parts, and parse them into the message digest function in sequential order. [Ull05]

The script presented in this example only creates a hash value of it self, and performance is not an issue. But with just small modifications this script could create a hash value for any selected file.

Next a Fileinputstream is created to the file example file it self.

```
/* this loop read 8192 bytes from file and updates hash, loop until eof */
DO FOREVER
    n=inStream~read(md) /* stream returns -1 if EOF */
    if n=-1 then leave  /* exit loop if EOF */
    messageD~update(md,0,n) /* update the hash value */
END
```

In the code block above data is read from the stream, and filled into the array with 8192 bytes at the time. When the whole file is read, -1 is returned an the the loop is terminated.

In the last line of the loop the *messageD* object calls the method *update*, and parse the byte array as an argument. The *update* method could be called a in definitive number of times until the end of file is reached.

After the script has finished the processing of the file, the hash it self is created using the *digest* method of the *messageD* object.

```
hash = messageD~digest()
```
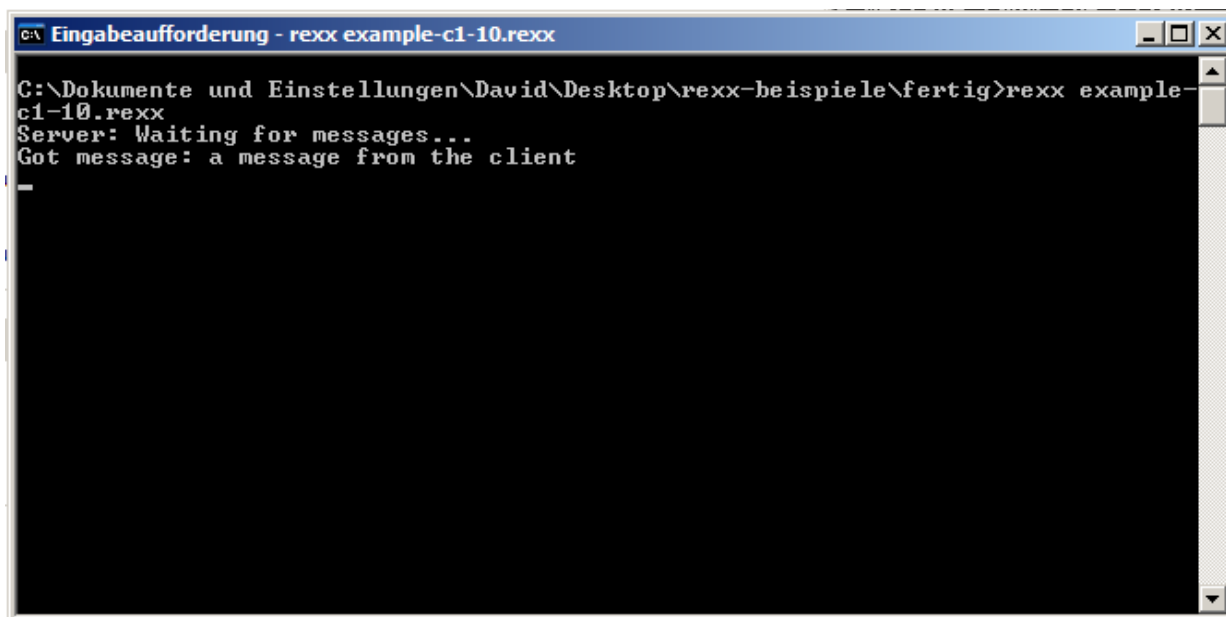
This method returns a byte array with the value of the hash. In order to get a nice looking hash value printed on the screen it is necessary to convert the byte array in to a string.

```
/* this loop convert the resulting hash from byte to hex */
DO until i>size
    b=.Byte~new(hash[i])~toString()
    hashPrint = hashPrint d2x( b ,2)
    i=i+1
END
```

In the code block above the resulting hash value is converted to string with the help of the the *toString* method of the *Byte* object. The string itself is again converted to a hexadecimal number using the ooRexx function *d2x* and added to the the value *hashPrint*. At the end of the script the *hashPrint* variable will contain the complete hash value.

3.1.10 Example C1-10 Java.net Server Classes

This example demonstrates how to create a running Server, which waits for messages sent by a client. This client is explained in the next example. To get a running server the script uses the Java.net classes

Figure 34: Output of example C1-10

```
srvSock=.bsf~new("java.net.ServerSocket", 8888)
/* import Java serversocket class an create a Instance on port 8888 */

SAY "Server: Waiting for messages..." /* output to console */
do while receivedData<>"e" /* create loop, which runs until an "e" is received */
        socket2client=srvSock~accept /* create an message accepting socket */
        inputFromClient=socket2client~getInputStream /* get input stream from socket */
        b=.bsf~bsf.createArray( 'byte.class', 2048)    /* create byte array */
        received=inputFromClient~read(b) /* read inputstream from client */
        strObject=.bsf~new('java.lang.String', b, 0, received)
                                                /* create String Object from received data */
        receivedData=strObject~toString /* extract String from StringObject */
        SAY "Got message:" receivedData /* output to console */
end
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 35: The code for example C1-10

3.1.10.1 Explanation

```
srvSock=.bsf~new("java.net.ServerSocket", 8888)
/* import Java serversocket class an create a Instance on port 8888 */
```

Open a new Serversocket on port 8888.

```
 socket2client=srvSock~accept
```

This line creates a message-accepting socket.

```
inputFromClient=socket2client~getInputStream /* get input stream from socket */
```

Read the Data incoming from the socket and store it as an InputStream in the variable "inputFromClient".

```
received=inputFromClient~read(b) /* read inputstream from client */
strObject=.bsf~new('java.lang.String', b, 0, received)
receivedData=strObject~toString /* extract String from StringObject */
```

These lines convert the InputStream to an String and assigns it to the "receivedData" variable.

```
SAY "Got message:" receivedData /* output to console */
```

Finally write it to the screen.

This example is simple and has no real difficulties, if the programmer is common with the usage of Java Net Classes.

3.1.11 Example C1-11 Java.net Classes for a simple client
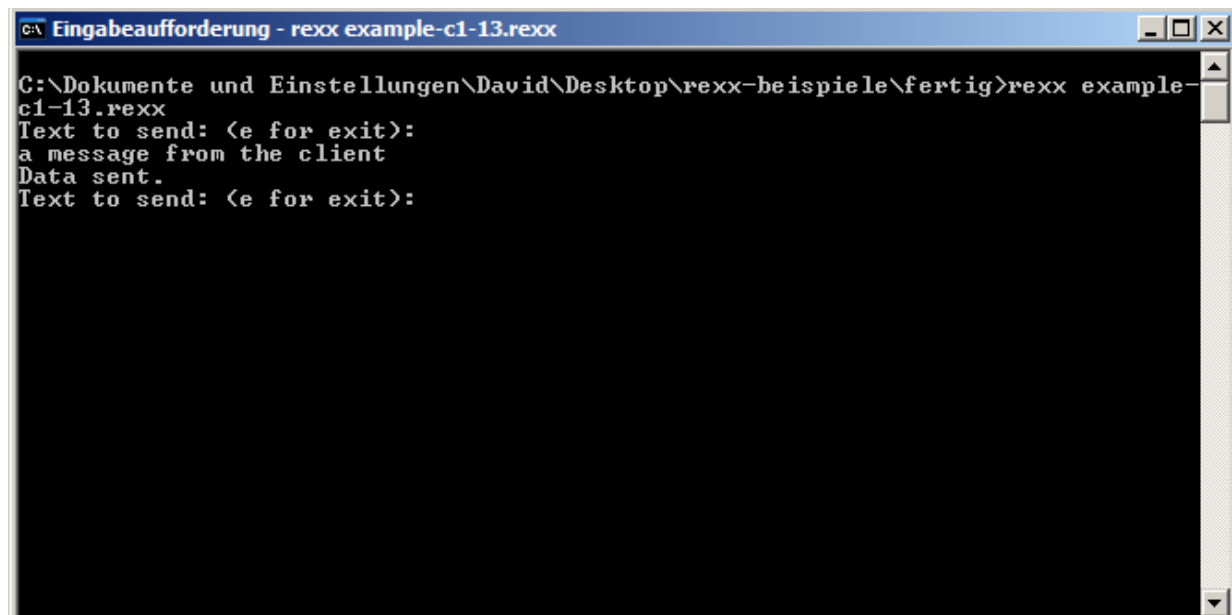
Figure 36: Output from example C1-11

This example demonstrates how to create a client, which sends messages to the server explained above. To create a client the script uses the Java.net classes

```
lh=.bsf~bsf.import('java.net.InetAddress') ~getLocalHost /* get the Localhost as Java InetAddress */

do while Datatosend<>"e"  /* create loop, which runs until an "e" is entered */
        socket2server=.bsf~new('java.net.Socket', lh, 8888) /* open socket to server */
        say "Text to send: (e for exit):" /* output to console */
        parse pull Datatosend /* get user input to send to server */
        str=.bsf_proxy~new(Datatosend) /* create new proxy */
        os=socket2server~getOutputStream /* get output stream */
        os~write(str~getBytes) /* write data to output stream */
        SAY "Data sent." /* output to console */
end
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 37: The code of example C1-11

3.1.11.1 Explanation

```
lh=.bsf~bsf.import('java.net.InetAddress') ~getLocalHost /* get the Localhost as Java InetAddress */
```

In this example, the server and the client are on the same computer. So it needs Java Inet-Address of the localhost. If you want to try this on different computer you have to use the "getbyAdress" or the "getbyName" method of the Java.net.InetAddress class.

```
do while Datatosend<>"e"
```

This line creates an loop, which ends when the client sends an "e".

```
socket2server=.bsf~new('java.net.Socket', lh, 8888)
```

Open a socket to the server using the above create Java InetAddress and the port number
8888 as arguments.

```
parse pull Datatosend /* get user input to send to server */
```

Read the user input. The automatic converting to uppercase ist avoided using the command "parse".

```
str=.bsf_proxy~new(Datatosend) /* create new proxy */
```

Converting the inputdata to an proxy.

```
os=socket2server~getOutputStream /* get output stream */
os~write(str~getBytes) /* write data to output stream */
```

Get the Outputstream of the socket to the server and write the data of the above created
proxy to it.

3.1.12 Example C1-12 – Drawing Charts

This example draws a pie-chart using the JFreechart library.

3.1.12.1 The JFreeChart Library

JFreeChart is a free, open source chart library written in Java. The JFreeChart project was started in the year 2000 by David Gilbert. Today up to 50.000 developers are using this chart library in their applications [JFree06]. JFreeChart is distributed under the terms of the GNU Lesser General Public License (LGPL), which, other than the GNU General Public License (GPL), also allows the use in non-free, commercial applications [LGPL06]. Furthermore JFreeChart provides the following features [JFree06]:

- a consistent, well documented API

- support of a wide range of chart types

- support of multiple output formats (images, Swing components, vector graphics)

- a flexible design which makes it easy to extend and suited for client sided applications as well as server sided ones.

```
/* import classes from the JFreeChart API */
.bsf~bsf.import("org.jfree.chart.ChartUtilities", "ChartUtilities")
.bsf~bsf.import("org.jfree.chart.ChartFactory", "ChartFactory")
.bsf~bsf.import("org.jfree.chart.ChartFrame", "ChartFrame")
.bsf~bsf.import("org.jfree.data.general.DefaultPieDataset", "DefaultPieDataset")

/* import Standard Java Classes */
.bsf~bsf.import("java.io.File","File")
.bsf~bsf.import("java.lang.Integer","Integer")

/* create a pie chart with four areas */
pieDataset = .DefaultPieDataset~new();
pieDataset~setValue("A", .Integer~new(75));
pieDataset~setValue("B", .Integer~new(10));
pieDataset~setValue("C", .Integer~new(10));
pieDataset~setValue("D", .Integer~new(5));

chart = .ChartFactory~createPieChart("A great Chart...", pieDataset, .true, .true, .false);

/* Define a Frame for the chart(using the pre-definde ChartFrame class)*/
chartFrame = .ChartFrame~new("Sample Chart", chart);
chartFrame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"))

/* set size + make frame visible  */
chartFrame~~setSize(300,400)~~setVisible(.true);

res = BSF("sleep", 60.00)  --wait for 60 seconds

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 38: The code for example c1-12

The screenshot shows the result of the Rexx code: a Swing Frame that displays a pie chart.
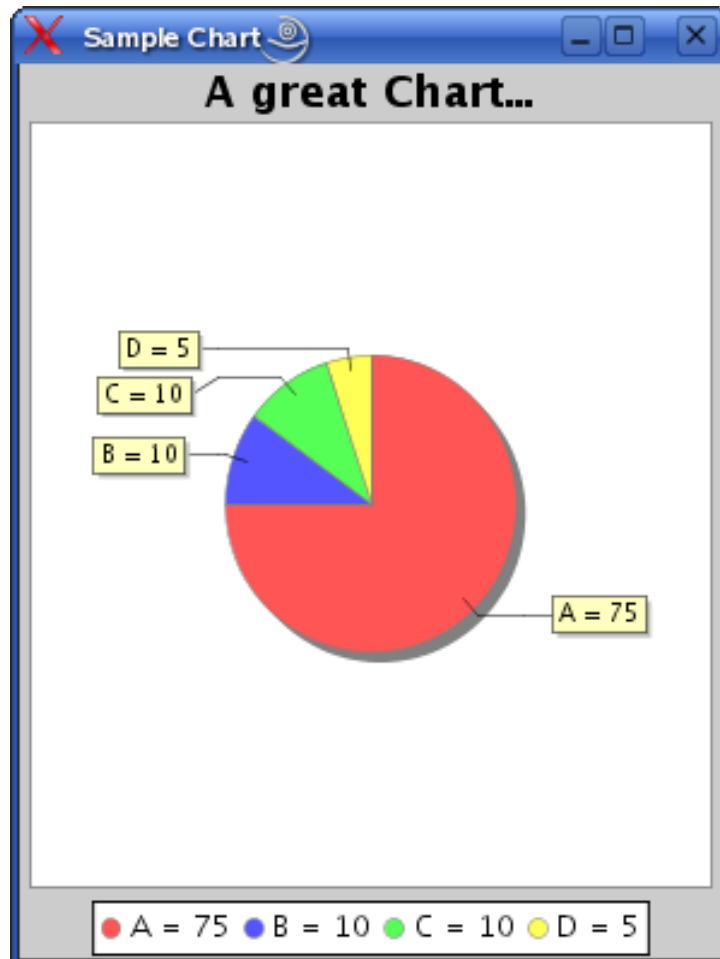
Figure 39: Pie Chart Frame

3.1.12.2 Explanation

After the import of the necessary Java classes a "PieDataset" object is created:

```
pieDataset = .DefaultPieDataset~new();
pieDataset~setValue("A", .Integer~new(75));
pieDataset~setValue("B", .Integer~new(10));
pieDataset~setValue("C", .Integer~new(10));
pieDataset~setValue("D", .Integer~new(5));
```

JFreeChart provides the class "DefaultPieDataset" for creating pie chart data. The object is initialized with some key-value pairs over the "setValue" function. There are different dataset interfaces for the other chart types. In the next step the ChartFactory class is used to create a chart object:

```
chart = .ChartFactory~createPieChart("A great Chart...", pieDataset, .true, .true, .false);
```

The ChartFactory provides methods to create all kinds of charts supported by JFreeChart. The method "createPieChart" expects an object implementing the "PieDataset" interface providing the chart data. The following parameters configure the chart to display a legend, tooltips and not to use URLs.

After initializing the chart, a "ChartFrame" object is created, with the chart as parameter. This class is provided by the JFreeChart API to display charts and extends the Swing class "JFrame".

Finally the chart is displayed by a call to the "setVisible" routine. A call to the BSF Function "sleep" at the end of the script prevents the program from exiting immediately. An

3.1.13 Example C1-13 Text-to-Speech with FreeTTS

This example implements the external Java library "FreeTTS" in order to create a text-to-speech application. FreeTTS is written entirely in Java, and because of this true multi platform applikations can be build in combination with ooRexx and BSF4Rexx.

In order to run this example the FreeTTS library need to be downloaded and installed. FreeTTS can be downloaded from this address: http://freetts.sourceforge.net/docs/index.php

A detailed description of how to install external Java libraries can be found in the beginning of this paper.

In this example a GUI is created with Swing, and displayed in the middle of the screen. The positioning is calculated dynamic, meaning that the resolution of the screen and the size of the frame is investigated and used in order to find the correct position.
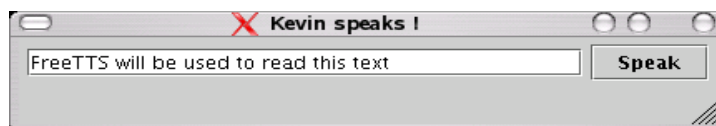
Figure 40: Screenshot of example C1-13

The GUI contains a field where the user can enter text. When the user push the enter button on the keyboard or push the speak button with the mouse, the text entered will be read out load using the FreeTTS library.

In the following example only the relevant code for free TTS and the code that position the window to the middle of the screen is explained. The creation of the GUI and the handling of events is described in detail for example C1-6.

One Problem that could specially on Unix systems occur, is that the sound device is occupied. An error message like this would be displayed:

```
LINE UNAVAILABLE: Format is PCM_SIGNED, 16000.0 Hz, 16 bit, mono, big-endian, audio data
```

To avoid this, all other applications using sound on the system should be closed.

```
/* import classes */
.bsf~bsf.import("com.sun.speech.freetts.Voice","JVoice")
.bsf~bsf.import("com.sun.speech.freetts.VoiceManager","JVoiceManager")
.bsf~bsf.import("com.sun.speech.freetts.audio.JavaClipAudioPlayer","JavaClipAudioPlayer")
.bsf~bsf.import("javax.swing.JFrame","JFrame")
.bsf~bsf.import("java.awt.FlowLayout","FlowLayout")
.bsf~bsf.import("javax.swing.JButton","JButton")
.bsf~bsf.import("javax.swing.JTextField","JTextField")

myvoice="kevin16"

/* create object using static method */
voiceManager = .JVoiceManager~getInstance();

/* get an instance of the voice engine */
voice = voiceManager~getVoice(myVoice)

/* activate the voice engine */
voice~allocate()

/* create GUI Objects */
frame=.JFrame~new("Kevin speaks !")
text=.JTextField~new()~~setColumns(30)
button=.JButton~new("Speak")

/* set the the frame to use the FlowLayout */
frame~getContentPane()~setLayout(.FlowLayout~new());

/* add the elements to frame */
frame~getContentPane()~~add(text)~~add(button)

/* Add eventhandling */
frame~setDefaultCloseOperation(.bsf~bsf.getStaticValue("javax.swing.JFrame","EXIT_ON_CLOSE"));
frame~bsf.addEventListener('window', 'windowClosing', 'exitApp = .true')
button~bsf.addEventListener('action', '', 'call read')
text~bsf.addEventListener('action', '', 'call read')

/* show the frame */
frame~~pack()~~setVisible(1)
/* this routine center the frame to the midle of the screen */
call centerFrame frame

/* do a loop and wait for the event handling to return with some text */
exitApp = .false
do until exitApp = .true
   a = bsf("pollEventText")      /* wait for an eventText to be sent     */
     interpret a                 /* execute as a Rexx program    */
end

/* unbind the voice engine */
voice~deallocate()

read:
    /* get the text from the JTextFrame and pass it on to the voice engine */
    voice~speak(text~getText())
    text~setText("")
return

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS

/* This routine center a frame to the middle of the screen */
::routine centerFrame
    use arg frame
    .bsf~bsf.import("java.awt.Toolkit","JToolkit")
    toolkit=.JToolkit~getDefaultToolkit()
    screenSize=toolkit~getScreenSize()-- returns a dimension object

    frame~pack()-- need to pack the frame before we get the frame size

    frameSize=frame~getSize()-- returns a dimension object

    X=(screenSize~width()-frameSize~width())/2
    Y=(screenSize~height()-frameSize~height())/2

    X=X%1
    Y=Y%1
    frame~setLocation(X,Y)       -- positions the frame on the screen
return
```

Figure 41: The code for example C1-13

3.1.13.1 Explanation

The following text explains the two main blocks of code in the example. First FreeTTS is used to output text as audio, then a routine is presented that positions a swing frame to the middle of the screen.

3.1.13.1.1 The Text-to-Speech Functionality

The code block below shows the FreeTTS code used in this example.

```
.bsf~bsf.import("com.sun.speech.freetts.Voice","JVoice")
.bsf~bsf.import("com.sun.speech.freetts.VoiceManager","JVoiceManager")
.bsf~bsf.import("com.sun.speech.freetts.audio.JavaClipAudioPlayer","JavaClipAudioPlayer")

myvoice="kevin16"

voiceManager = .JVoiceManager~getInstance();
voice = voiceManager~getVoice(myVoice)
voice~allocate()

vvoice~speak ("hello")

voice~deallocate()
```

First the required classes are imported, then the name of the voice to use is written into the variable *myvoice*. FreeTTS support several voices in different qualities, and kevin16 is one of them.

Next the voice manager is created using the static method *getInstance* from the class *com.sun.speech.freetts.VoiceManager*, that is given the name *JvoiceManager* in this example.

Then the selected voice is loaded an allocated, and the command *voice~speak()* can be called a number of times with an ooRexx variable as argument. The content of the variable will be parsed to the voice engine, that will generate the audio.

At the end of the script the method *deallocate* of the object *voice* is called in order to free resources hold by the FreeTTS engine.

3.1.13.1.2 Position a Frame to the Middle of the Screen

In this example a routine is presented that dynamically positions a frame to the middle of the screen. Dynamically means that the size of the frame and the resolution of the screen is used in order to find the exact middle position for the frame.

The code block below shows the routine used in this example. This method is general an could easy be copied into all other types of BSF4Rexx scripts that requires this functionality.

```
::routine centerFrame
    use arg frame
    .bsf~bsf.import("java.awt.Toolkit","JToolkit")
    toolkit=.JToolkit~getDefaultToolkit()
    screenSize=toolkit~getScreenSize()-- returns a dimension object

    frame~pack()-- need to pack the frame before we get the frame size

    frameSize=frame~getSize()-- returns a dimension object

    X=(screenSize~width()-frameSize~width())/2
    Y=(screenSize~height()-frameSize~height())/2

    X=X%1
    Y=Y%1
    frame~setLocation(X,Y)      -- positions the frame on the screen
return
```

First the *class java.awt.Toolkit* is imported and given the name *JToolkit*. Normally the import statement will be at the top of the script, but in order for developers to easy test this code in other BSF4Rexx scripts, the import statement is placed inside of the routine.

Next a *toolkit* object is created and the resolution of the screen is investigated using the *getScreenSize* method.

This method returns an object of type dimension. To get the the number of horizontal and vertical pixels the methods *width* and *hight* have to be used.

Then the size of the frame is investigated with the *getSize* method of the *frame* object. This method also return an object of type dimension.

After the new and centered X and Y value is calculated for the upper left position of the frame, the values are converted to integers, and the *frame* is set to the new location with the method *setLocation*.

3.1.14 Example C1-14 – Playing MP3's

This example demonstrates how to use the JLayer library to play MP3 files from Rexx.

3.1.14.1 The JLayer Library

The JLayer library provides Java classes that allow playing MP3 encoded music files. The JLayer project was founded in 1999 and is completely volunteer driven. JLayer serves as a base for multiple Java-based MP3 player applications, among others the jlGui music player [Jlayer06]. JLayer, as well as JLayerME for J2ME, are licensed under LGPL (see also example C1-11, p. 62)

```
/* import Player Class from the JLayer API */
.bsf~bsf.import("javazoom.jl.player.Player", "Player")

/* import standart Java Classes */
.bsf~bsf.import("java.io.FileInputStream", "FileInputStream")
.bsf~bsf.import("java.io.BufferedInputStream", "BufferedInputStream")

/* create Buffered input stream */
fis = .FileInputStream~new("12.mp3")
bis = .BufferedInputStream~new(fis)

/* create a player Object and play the MP3 file */
player = .Player~new(bis)
player~play();


/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 42: The code for example c1-14

3.1.14.2 Explanation

First of all the necessary Java classes for the program need to be loaded. This is done by the "bsf.import" function of the BSF class. Then a Java "FileInputStream" is created with the location of a valid ".mp3" file as parameter. The "FileInputStream" object is passed to a new "BufferedInputStream" that serves as input for the constructor of "Player". The JLayer class "Player" already implements a simple player.

```
player~play();
```

By a call to the player objects "play" method, the player starts playing the .mp3 file.

JLayer also offers more advanced options for playing audio files like an equalizer class, classes for managing audio devices and decoding methods. These classes can be found in the JLayer API documentation and used in a similar way (http://www.javazoom.net/javalayer/docs/docs1.0/index.html).

3.1.15 Example C1-15 Parse XML with JDOM

This example uses JDOM to parse an Extensible Markup Language (XML) file in order to read out selected values.

JDOM is an external Java library for accessing, manipulating and output XML data.
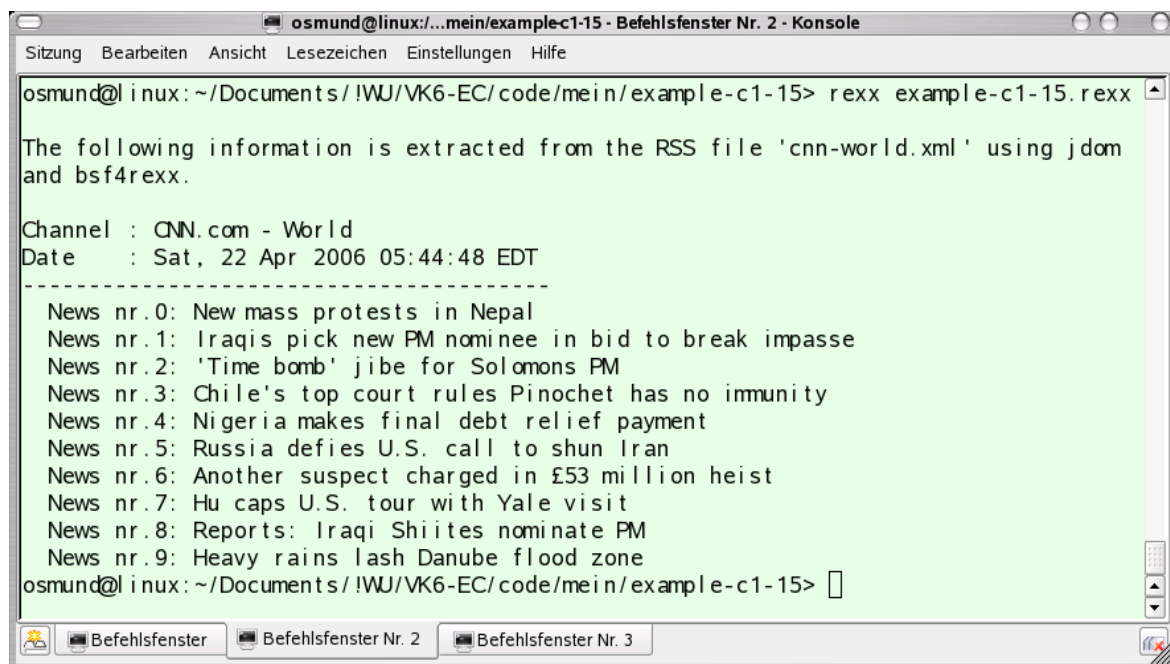
In order to run this example the JDOM library needs to be downloaded and installed. JDOM can be downloaded from this address: http://www.jdom.org/

A detailed description of how to install external Java libraries can be found in the beginning of this paper.

This example is delivered with an Really Simple Syndication (RSS) file with the name "cnn-world.xml". A RSS file is a XML file specialized for delivering news content.

The script reads the RSS file and return selected information from it.

The following screen shot shows the output from the script.

```
osmund@linux:~/Documents/!WU/VK6-EC/code/mein/example-c1-15> rexx example-c1-15.rexx

The following information is extracted from the RSS file 'cnn-world.xml' using jdom
and bsf4rexx.

Channel : CNN.com - World
Date    : Sat, 22 Apr 2006 05:44:48 EDT
----------------------------------------
  News nr.0: New mass protests in Nepal
  News nr.1: Iraqis pick new PM nominee in bid to break impasse
  News nr.2: 'Time bomb' jibe for Solomons PM
  News nr.3: Chile's top court rules Pinochet has no immunity
  News nr.4: Nigeria makes final debt relief payment
  News nr.5: Russia defies U.S. call to shun Iran
  News nr.6: Another suspect charged in £53 million heist
  News nr.7: Hu caps U.S. tour with Yale visit
  News nr.8: Reports: Iraqi Shiites nominate PM
  News nr.9: Heavy rains lash Danube flood zone
osmund@linux:~/Documents/!WU/VK6-EC/code/mein/example-c1-15>
```

Figure 43: Output from example C1-15

```
rssFile = "cnn_world.xml"
xpathItems ="/rss/channel/item/title"
xpathChannelTitle ="/rss/channel/title"
xpathChannelDate ="/rss/channel/pubDate"

/* import classes */
.bsf~bsf.import("org.jdom.Document","Document")
.bsf~bsf.import("org.jdom.input.SAXBuilder","SAXBuilder")
.bsf~bsf.import("java.io.File","File")
.bsf~bsf.import("org.jdom.xpath.XPath","XPath")

/* use sax to read file into document tree. */
builder = .SAXBuilder~new()

/* create DOM-modell */
doc = builder~build(.File~new(rssFile))

/* use XPpath to search the DOM tree. Return a list */
titleElement = .XPath~selectNodes(doc, xpathChannelTitle)
pubDateElement = .XPath~selectNodes(doc, xpathChannelDate)
itemElements = .XPath~selectNodes(doc, xpathItems)

/* Print out the channel name and date */
say -"The following information is extracted from the RSS file 'cnn-world.xml' using jdom andBSF4Rexx."
say
say "Channel :" titleElement~get(0)~getText()
say "Date    :" pubDateElement~get(0)~getText()
say "------------------------------------"

/* loop over all items in the rss feed, print out the item title */
i = 0
do until i = itemElements~size()
    say "  News nr."i": "itemElements~get(i)~getText()
    i = i + 1
end

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 44: The code for example C1-15

3.1.15.1 Explanation

In the first line of the script the variable *rssFile* is set to the name of the RSS file to load. Then three variables are set with three different XPath expressions.

XML Path Language (XPath) is a query language for XML data. XPath is developed from W3C-Consortium and exist currently in version 1.0. [Wiki06-3] With XPath a developer can address s specific element in a XML structure, or a set of elements.

After the required libraries are loaded, an object of the type *SAXBuilder* is created. This object has a method build, that takes a file as an argument, an loads the file into an internal memory structure. After this is done a object is returned into the variable *doc*.

With the *selectNode* method from the *doc* object, a Xpath expression can be used to query the internal memory structure.

```
/* use XPpath to search the DOM tree. Return a list */
titleElement = .XPath~selectNodes(doc, xpathChannelTitle)
pubDateElement = .XPath~selectNodes(doc, xpathChannelDate)
itemElements = .XPath~selectNodes(doc, xpathItems)
```

First the title of the channel is queried with the XPath expression created on the top of the script. Then the date of the channel is queried, and at the end all the news items.

The method *selectNodes* returns an object with the reference to the queried element. This reference object can contain one or more elements.

In order to get the text of the node, it is necessary to first use the method *get* to get correct element, and then to apply the method *getText* to get the text of it, and not the object it self. As demonstrated in the code under the first element is selected and then the method *getText* is applied to get the text of the element.

```
say "Channel :" titleElement~get(0)~getText()
```
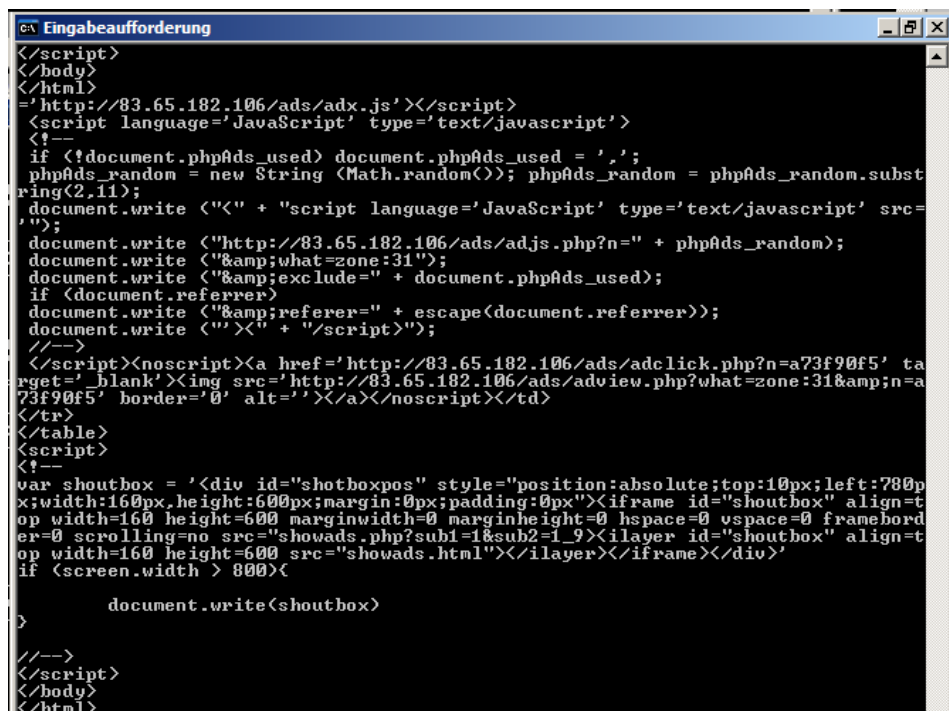
Almost the same is happening in the code block under. The object *itemElements* contains all news elements in the RSS file. A loop is created in order to get the text of all elements, and print them to the screen.

```
/* loop over all items in the rss feed, print out the item title */
i = 0
do until i = itemElements~size()
    say "  News nr."i": "itemElements~get(i)~getText()
    i = i + 1
end
```

The method *size* of object *itemElements* returns all the elements in the *itemElements* object.

3.1.16 Example C1-16 Java.net Classes for sending an HTTP/GET Request

This example demonstrates how to create a client, which sends a HTTP/GET Request to an server and print the response of the server on the screen. To create a client the script uses the Java.net classes.

Figure 45: Output from example C1-16

```
lh=.bsf~bsf.import('java.net.InetAddress')~getByName("http://www.bundesliga.at")
                                        /* Get the Java Inetaddress of the Host */
socket2server=.bsf~new('java.net.Socket', lh, 80) /* open socket to server */
out = socket2server~getOutputStream() /* get output stream */
newLine = "0a0a"x
str=.bsf_proxy~new("GET /bewerbe/index.php?&sub1=1&sub2=1_9&sub3=1_9_2 HTTP/1.0" || newLine || - newLine )
/* create new proxy */
out~write(str~getBytes) /* write data to output stream */
b=.bsf~bsf.createArray( 'byte.class' , 200001)  /* create byte array */
len=0
Datafromserver=''

do while  len>=0 /* create loop, which runs until no data is sent anymore */
        len = socket2server~getInputStream~read(b)/* read inputstream */
        if len>0 then
                strObject=.bsf~new('java.lang.String', b, 0, len)
                                        /* create String Object from received data */
        Datafromserver = Datafromserver || strObject~toString
                /* extract String from StringObject and add it to the already received data */
end
SAY "Daten empfangen:" || Datafromserver /* output to console */
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 46: The code of example C1-16

3.1.16.1 Explanation

```
lh=.bsf~bsf.import('java.net.InetAddress')~getByName("http://www.bundesliga.at")
```

The first line gets the Java InetAddress of the url "http://www.bundesliga.at"

```
socket2server=.bsf~new('java.net.Socket', lh, 80) /* open socket to server */
out = socket2server~getOutputStream() /* get output stream */
```

Create a socket to the server on port 80 and get the outputstream of the socket.

```
newLine = "0a0a"x
```

The "x" after the string indicates that this is an hexadecimal string. A linebreak is defined as "0a0a" in hexadecimal format.

```
str=.bsf_proxy~new("GET /bewerbe/index.php?&sub1=1&sub2=1_9&sub3=1_9_2 HTTP/1.0" || newLine || - newLine )
                                                          /* create new proxy */
out~write(str~getBytes) /* write data to output stream */
```

Send an standard HTTP/GET Request to the server.

```
b=.bsf~bsf.createArray( 'byte.class' , 200001)  /* create byte array */
```

Create a new byte array with the size 200001, which will contain the response of the server.

```
do while  len>=0
```

This line starts a loop. Which runs until no data is received anymore.

```
len = socket2server~getInputStream~read(b)/* read inputstream */
```

Get the received Data and store it in the byte array "b". The length of the received Data is assigned to the variable "len".

```
if len>0 then
            strObject=.bsf~new('java.lang.String', b, 0, len)
                                      /* create String Object from received data */
        Datafromserver = Datafromserver || strObject~toString
```

Convert the received Data to a string and add it to the already received Data.

3.1.17 Example C1-17 3D Graphics

This example demonstrates how to invoke the Java 3D API with BSF4Rexx. Since this API includes many classes and provides complex functionality, only a "proof of concept" can be provided in the scope of this paper.

3.1.17.1 Java 3D

The Java 3D project started in 1996, when Intel, Silicon Graphics, Apple and Sun decided to collaborate in creating a retained mode scene graph API. The first beta version was released in March 1998. In 2004 Java 3D became a community source project, developed by Sun and volunteers [wiki3D06]. The Java 3D API provides a collection of high-level constructs for creating and manipulating 3D geometry and structures for rendering that geometry [J3D00, p.1-1].

```
/* import classes from the Java 3D API */
.bsf~bsf.import("com.sun.j3d.utils.universe.SimpleUniverse", "SimpleUniverse")
.bsf~bsf.import("com.sun.j3d.utils.geometry.ColorCube", "ColorCube")
.bsf~bsf.import("com.sun.j3d.utils.geometry.Sphere", "Sphere")
.bsf~bsf.import("javax.media.j3d.BranchGroup", "BranchGroup")
.bsf~bsf.import("javax.media.j3d.Transform3D", "Transform3D")
.bsf~bsf.import("javax.media.j3d.TransformGroup", "TransformGroup")
.bsf~bsf.import("java.lang.Math", "Math")

/* create a 'SimpleUniverse' that will contain our objects */
universe = .SimpleUniverse~new();


/* create a transformation node */
rotate = .Transform3D~new
/* rotate */
rotate~rotY(.Math~PI/5.0)

/*attach the transformation node to a group */
objRotate = .TransformGroup~new(rotate)

/*add a cube object */
objRotate~addChild(.ColorCube~new(0.4))

/* create a data structure to contain some objects */
/* and add the rotation group */
branchGroup = .BranchGroup~new()~~addChild(objRotate);

/* set the camera "eye" */
universe~getViewingPlatform()~setNominalViewingTransform();

/* add the objects to the universe*/
universe~addBranchGraph(branchGroup)

/* wait 10 seconds */
 res = BSF("sleep", 10.00)


/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 47: The code of example C1-17 (based on [J3D00, p. 1-21]).
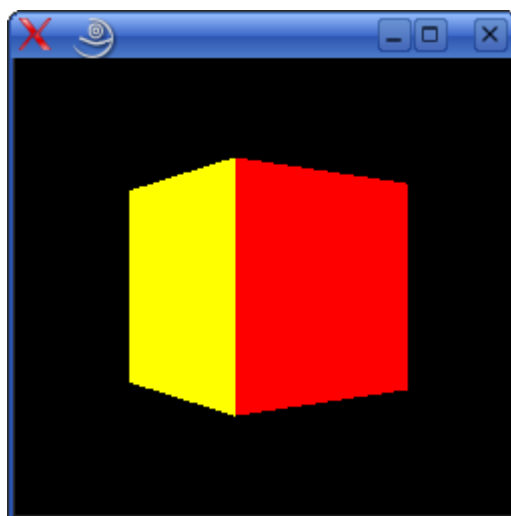
Figure 48: Output of example C1-17.

3.1.17.2 Explanation

After the import of the necessary classes from the Java 3D API, a *SimpleUniverse* object is created. This is the three dimensional space in which objects can be placed. This class automates some steps that are needed when creating a scene graph [see also J3D00, p. 1-9].

```
universe = .SimpleUniverse~new();
```

In the next step, the transformation node *rotation* is created. This node represents a single transformation step. The methods *rotX* and *rotY* allow rotation along X and Y axis. In this case a rotation along the Y axis is performed.

```
rotate~rotY(.Math~PI/5.0)
```

The transformation node is passed to the constructor of a *TransformGroup*. This is an object, that can position, orient, and scale all of its children [see also J3D00, p. 1-21ff].

```
objRotate = .TransformGroup~new(rotate)
```

In the next step a cube is also added to the TransformGroup as a child. The transformation specified in the *Transform3D* object will be applied to this cube.

```
objRotate~addChild(.ColorCube~new(0.4))
```

Next, a *BranchGroup* object is created. A *BranchGroup* object is the root of a subgraph (from the top-level scene graph) [J3D00, p. 1-7]:

```
branchGroup = .BranchGroup~new()~~addChild(objRotate);
```

In the same line the group node with the cube is attached to the graph.

The *SimpleUniverse* object has a *ViewingPlatform* member object. With a call to its *setNominalViewingTransform* method, the position of the viewer ("eye" position) is set to a centered position 2.41 meters in front of the scene [J3D00, p. 1-12]

```
universe~getViewingPlatform()~setNominalViewingTransform();
```

Finally the branch group (with the sphere and the cube) is added to the scene.

The result can be seen in figure 34 – a cube rotated along the Y-axis.

Java 3D is a complex API (over 100 core classes). Therefore many classes are involved in more complex examples. This requires a good knowledge of the API, since no helper class (like UNO.CLS, see 3.1) is available.

3.1.18 Example C1-18 Read ID3 Tags from MP3 files

This example uses an external Java library called ID3 for reading meta data from mp3 files. The meta data could be e.g. the name of the artist or the album.

The user needs to start the script with the name of a mp3 file as an argument. The script then uses the ID3 library to output some essential meta data for the file.

In order to run this example the ID3 library need to be downloaded and installed. ID3 can be downloaded from this address: http://www.ueberdosis.de/java/id3.html

A detailed description of how to install external Java libraries can be found in the beginning of this paper.

```
/* check if the user provided an argument */
parse arg filename
if filename = "" then
do
    say "Please use the name of a mp3 file as first argument"
    exit
end

/* import classes */
.bsf~bsf.import("java.io.RandomAccessFile","RAFile")
.bsf~bsf.import("de.ueberdosis.mp3info.ID3Reader","ID3Reader")
.bsf~bsf.import("de.ueberdosis.util.OutputCtr","OutputCtr")

/* Instruct the ID3 library to print no debug messages on the screen*/
.OutputCtr~setLevel(0)

/* open a Random Access File for reading */
/* use the static method readTag in ID3Reader to get a ID3tag object */
tag = .ID3Reader~readTag(.RAFile~new(filename,"r"))

/* use the ID3tag object to get some information */
say "Extract ID tags from file:" filename
say "   Artist  =" tag~getArtist()
say "   Year    =" tag~getYear()
say "   ALbum   =" tag~getAlbum()
say "   Title   =" tag~getTitle()
say "   Title   =" tag~getGenreS()
say "   Comment =" tag~getComment()

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 49: The code for example C1-18

3.1.18.1 Explanation

After the necessary classes are loaded, the log level of the ID3 level is set using the static method *setLevel* from the *de.ueberdosis.util.OutputCtr* class.

```
.OutputCtr~setLevel(0)
```

By default the ID3 library writes quite much information to standard out. This is information that might be interesting, specially for debugging purposes, but in a normal command line

script this might be a disturbance. To completely disable all log messages the log level has to be set to "0".

The usage of the ID3 library is quite easy. First the mp3 file has to loaded. This is done using the static method *readTag* from the *de.ueberdosis.mp3info.ID3Reader* class, as shown bellow.

```
tag = .ID3Reader~readTag(.RAFile~new(filename,"r"))
```

This method requires a RandomAccessFile, and not just a simple File.

In order to read the actual meta data from the file, a get method is used, like the one displayed in the box under.

```
say "   Artist  =" tag~getArtist()
```

The ID3 library can also be used for writing id3 meta data into mp3 files. More information about this can be found at the ID3 homepage and in the API documentation.[8]

---

[8]   http://www.ueberdosis.de/java/id3/doc/index.html

3.1.19 Example C1-19 Java.calender Classes for Creating a Calendar

This example demonstrates how to create a calender of a whole year using the Java Calender Classes. The user is asked for a year and the program will print the calendar for this year.

Figure 50:
Output from
example C1-19,
part 1

Figure 51:
Output from
example C1-
19, part 2

Figure 52:
Output from
example C1-
19, part 3

```
/*-----------------------------main program---------------------------------*/
.bsf~bsf.import("java.util.Calendar","Calendar") /* import Calender class from Java */
.bsf~bsf.import("java.util.GregorianCalendar","GCalendar")
                                         /* import GregorianCalender class from Java */
/* set array which contains the names of the months */
month. = ''
month.0=January
month.1=February
month.2=March
month.3=April
month.4=May
month.5=June
month.6=July
month.7= August
month.8=September
month.9=October
month.10=November
month.11=December
/* set array which contains the number of days for each month */
dom.=''
dom.0=31
dom.1=28
dom.2=31
dom.3=30
dom.4=31
dom.5=30
dom.6=31
dom.7=31
dom.8=30
dom.9=31
dom.10=30
dom.11=31

say "Bitte das Jahr eingeben: "; /* output to console */
parse pull yy; /* input from console */
do mm = 0 to 11 by 1 /* call the function "print" for each of the twelve months of a year */
        call print mm, yy;
end
exit
print :
        parse arg mm, yy
                say ""; /* output to console */
                say "--------------------"/* output to console */
                say month.mm "  " yy /* output to console */
                say "--------------------"/* output to console */
                say "So Mo Tu We Th Fr Sa"/* output to console */
                daysinMonth=dom.mm /* get the number of days for the current month */
                GC = .GCalendar~new(yy,mm,1)
                /* create a new GregorianCalender Object for the first day in the current month */
                leadGap = GC~get(GC~DAY_OF_WEEK)-1
/* get the name of the day e.g. "Monday" in the form of 0(Sunday) to 6(Saturday). This information*/
/* is needed to know how much space we have to leave free in the first line of a month */
                if GC~isLeapYear(GC~get(GC~Year))=1 & mm=1 then
                                        /* if it is a leapyear then the February has 29 days */
                        daysinMonth = daysinMonth + 1
                output = ""
                do i = 1 to leadGap by 1
                        output = output || "   "
                        /* write blanks to get the number of a day under the right name (e.g. "Su") */
                end
                do i = 1 to daysinMonth by 1 /* write the number of each day */
                        if i<10 then
                                output = output || " "
                        output = output || i
                        if (leadGap+i)//7==0 | i==daysinMonth then
                                        /* make a line-break if it reaches the end of a week */
                                do
                                        say output
                                        output = ""
                                end
                        else
                                output = output || " "
                end
return
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Figure 53: The code of example C1-19

3.1.19.1 Explanation

The first value of the following arrays has the index 0. This can be confusing because the first month in the year has the index 0 and not 1 as in calendars.

```
month. = ''
month.0=January
month.1=February
month.2=March
```

```
month.3=April
month.4=May
month.5=June
month.6=July
month.7=August
month.8=September
month.9=October
month.10=November
month.11=December
```

Create an array containing the names of all the months of a year.

```
dom.=''
dom.0=31
dom.1=28
dom.2=31
dom.3=30
dom.4=31
dom.5=30
dom.6=31
dom.7=31
dom.8=30
dom.9=31
dom.10=30
dom.11=31
```

"dom" stands for "days of month". This array contains the number of days for each month.

e.g. January has 31 days. February has 28 days.

```
do mm = 0 to 11 by 1 /* call the function "print" for each of the twelve months of a year */
      call print mm, yy;
end
```

For each of the months the procedure "print", which is defined afterwards, gets executed with the number of the month and the year.

```
print :
      parse arg mm, yy
```

With this line the definition of the procedure print begins. The second line receives the arguments, which are needed for the execution of the procedure.

```
GC = .GCalendar~new(yy,mm,1)
```

Create a GregorianCalender instance for the currently processed month.

```
leadGap = GC~get(GC~DAY_OF_WEEK)-1
```

This line assigns the amount of space we need to leave free in the first line of each month to get the first number under the right day-name.

```
if GC~isLeapYear(GC~get(GC~Year))=1 & mm=1 then
                                /* if it is a leapyear then the February has 29 days */
                daysinMonth = daysinMonth + 1
```

These lines adds a day to the number of days of February if the year is a leap year.

```
if i<10 then
      output = output || " "
```

If the number of the day is lower than 10, a space have to be added in front of it to provide each number with a length of two to get a pretty print on the screen.

```
if (leadGap+i)//7==0 | i==daysinMonth then /* make a line-break if it reaches the end of a week */
                do
                        say output
                        output = ""
                end
```

After each week or at the end of a month, the program has to print the current week and start a new line.

While working with test classes you have to be aware that Java starts to count with "0". So January doesn't is the month with the number "1", but has the number 0 and so on. Additionally in these classes a week starts with "Sunday" and not with "Monday", what is common in most European Countries.

3.1.20 Example C1-20 JDBC

This example uses HSQLDB to create a database, and read an write data from it. HSQLDB is a SQL database that is completely programmed in Java, and hence it is multi platform. This is also the same database OpenOffice.org uses as its database engine in the OpenOffice.org Base component. [Hsq06]

Java Database Connectivity (JDBC) is a Java standard for how databases and Java programs interface.

The HSQLDB-JDBC driver can be used as a standalone database, which is demonstrated in this example. However scripts in this examples are so general, that other SQL-JDBC drivers could be used with just minor changes to the code.

In order to run this example the HSQLDB-JDBC driver needs to be downloaded and installed. HSQLDB can be downloaded from this address: http://hsqldb.org/

A detailed description of how to install external Java libraries can be found in the beginning of this paper.

This example contains the following two scripts.

- createDB.Rexx

- logDB.Rexx.

The "createDB.Rexx" script creates a database using the HSQLDB-JDBC driver, and the "logDB.Rexx" asks the user for some values to insert into the database. The existing data from the database is also displayed to the user.

```rexx
say "This script will create a new HSQLDB database in the directory ./data"
say "Do you want to continue ? (y/n)"
pull answer
if answer <> Y then exit

/* catch all errors */
signal on error
signal on failure
signal on syntax
signal on novalue

/* set some variabels used to connect to the database */
dbJdbcDriver = "org.hsqldb.jdbcDriver"
dbConnectionString = "jdbc:hsqldb:file:./data/"
dbName = "sa" /* the db user name */
dbPassword = "" /* the password */

sqlCreateTable = "CREATE TABLE log(id INTEGER IDENTITY,name VARCHAR,log VARCHAR)"

/* import classes */
.bsf~bsf.import("java.sql.DriverManager","DriverManager")

/* load the jdbc driver */
.BSF4Rexx~system.class~forName(dbJdbcDriver)

/* connect to database, return connection object */
con = .DriverManager~getConnection(dbConnectionString, dbName, dbPassword)

/* get a statement object from connection object */
stmt = con~createStatement()

/* This is a command to create a new table.. */
stmt~executeUpdate(sqlCreateTable)

/* Fille the new table with a row */
stmt~executeUpdate("INSERT INTO log(name,log) VALUES('Rexx','This is the first row.')")

/* close connection to database */
stmt~execute("SHUTDOWN");
stmt~close()
con~close()

say "Jippi ! HSQLDB-Database is created in ./data."
say "You are ready for the next step. Execute 'Rexx logDB.Rexx'"
exit

/* if an error occurs this code block will be executed */
error: failure: syntax: novalue:
say
say " ---------- An error occourd ----------"
say
say "My guesses are :"
say
say "- Java can't find the jdbc driver"
say "   - solution : copy hsqldb.jar to {your-java-installation-dir}/lib/ext/"
say
say "- you already have a database in ./data"
say "   - solution : delete the directory ./data"
say
say "- you have only read and no write permissions in the current directory"
say "     - you need this"
say
say "This is some more information about the error:"
say condition("C") "line #" sigl": ["sourceline(sigl)"]"
co=condition("O")    -- get the condition object
say "further information using the condition object:"
do idx over co
  say "   co~"idx"=["co~entry(idx)"]"
end
exit

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 54: The code for example C1-20, createDB.Rexx

3.1.20.1 Explanation – createDB.Rexx

The "logDB.Rexx" script could be executed several times, but the "createDB.Rexx" should only be executed once. If the "createDB.Rexx" is executed more than once, an error will occur. Normally when it comes to an error, the script will terminate where the error happened in the script, and all subsequent lines of code will not be executed. This can be

avoided by catching the error with the ooRexx command "signal on", as demonstrated in "createDB.Rexx". Explanation - createDB.Rexx

First the script asks the user if he would like to create a new database. If the user confirms this the script proceed. If not, the script exits at this point.

Then a series of *"signal on xxx"* is executed. This is code that instructs the ooRexx compiler to not directly abort on errors, but instead look for the code in the box under, and continue the execution of the script there in case of an error.

```
error: failure: syntax: novalue:
```

BSF4Rexx do not have the possibility to catch what type of error a Java objects throws, so the type of error is more or less unknown. But of course the uncertainty depends on how much code is placed between the *signal on* and the *error:* statements.

Next the *dbConnectionString* variable is set to the string "*jdb:hsqldb:file:./data/*". This string is later used to initialize the connection to the database. If the database does not already exist in the ./data/ directory, it will be created. If the script is changed to access another database, this connection string needs to be changed in order to reflect the use of different jdbc driver.

The user name and the password of the database also set in their respective variables and use later in the script.

After the necessary import statements, and the database driver is loaded, the connection to the database is made, as displayed on the box below.

```
/* connect to database, return connection object */
con = .DriverManager~getConnection(dbConnectionString, dbName, dbPassword)

/* get a statement object from connection object */
stmt = con~createStatement()
```

The method *createStatement* is used on the connection Object, and the returning object is the actually object used for sending Structured Query Language (SQL) statements to the server.

When modifying data the *executeUpdate* method is used, as the case is for *"CREATE TABLE ..."* and *"INSERT INTO ..."* SQL statements, as demonstrated in the code bellow.

```
/* This is a command to create a new table.. */
stmt~executeUpdate(sqlCreateTable)
```

If data is search for as the case is for a "SELECT .." statement, the method *executeQuery* is used instead.

After the the table is created an a row is insert into it using the "INSERT INTO ..." SQL statement, the database is shutdown.

```
/* close connection to database */
stmt~execute("SHUTDOWN");
```

Normally this is not necessary, because the server is supposed to serve several clients, and it would be no point of shutting it down when the client program ends. But in this case there is only one client, and the Java hsqldb engine will anyway terminate when the script terminates. Sending the shutdown command to the server makes sure that the server close it files and saves its data before exiting. Without the shutdown command, data in the database might become inconsistent. After this the statement and connection objects are closed. If no errors occurred, the script will terminate at this point. If errors occur the code after *"error: failure: syntax: novalue:"* will be executed.

```rexx
/* set some variabels used to connect to the database */
dbJdbcDriver = "org.hsqldb.jdbcDriver"
dbConnectionString = "jdbc:hsqldb:file:./data/"
dbName = sa
dbPassword = ""

sqlCreateTable = "CREATE TABLE log(id INTEGER IDENTITY,name VARCHAR,log VARCHAR)"

/* catch all errors */
signal on error
signal on failure
signal on syntax
/* signal on novalue */

/* import classes */
.bsf~bsf.import("java.sql.DriverManager","DriverManager")

/* load the jdbc driver */
.BSF4Rexx~system.class~forName(dbJdbcDriver)

/* connect to database, return connection object */
con = .DriverManager~getConnection(dbConnectionString, dbName, dbPassword)

/* get a statement object from connection object */
stmt = con~createStatement()

/* loop : ask the user for name and log data and store it in the db */
do until anotherTime = n
    say
    /* retrive row from the db */
    result = stmt~executeQuery("SELECT * FROM log")
    /* set the pointer to the first row */
    moreRows =result~next()
    do until moreRows = 0
        say "Log nr" result~getString("id") "from user" result~getString("name")" :" result~getString(log)
        /* get the next row or 0 if no more rows */
        moreRows =result~next()
    end

    say
    say "What is you name ?"
    parse pull name

    say "What do you want to log ?"
    parse pull logText

    /* write a new row into the table */
    stmt~executeUpdate("INSERT INTO log(name, log) VALUES('"name"','"logText"')")

    say "Write another row in the log ? (y/n)"
    pull anotherTime
end

/* close connection to database */
stmt~execute("SHUTDOWN");  /* this is hsqldb specific, (write result to disk) */
stmt~close()
con~close()
exit

/* if an error occurs this code block will be executed */
error: failure: syntax: novalue:
say
say " ---------- An error occurred ----------"
say
say "My guesses are :"
say
say "- Java can't find the jdbc driver"
say "  - solution : copy hsqldb.jar to {your-java-installation-dir}/lib/ext/"
say
say "   - you don't have a HSQLDB in the ./data directory"
say "          - solution : execute the createDB.Rexx script"
say
say "This is some more information about the error:"
say condition("C") "line #" sigl": ["sourceline(sigl)"]"
co=condition("O")    -- get the condition object
say "further information using the condition object:"
do idx over co
  say "    co~"idx"=["co~entry(idx)"]"
end
exit

/* make oo-like BSF4Rexx support available */
::requires BSF.CLS
```

Figure 55: The code for example C1-20, logDB.Rexx, part 1.

3.1.20.2 Explanation - logDB.Rexx.

The JDBC specific code in the "logDB.Rexx" is almost identical to the one in "createDB.Rexx". The only difference is that the "logDB.Rexx" at one point reads data from the database.

```
    /* retrive row from the db */
    result = stmt~executeQuery("SELECT * FROM log")
    /* set the pointer to the first row */
    moreRows =result~next()
    do until moreRows = 0
say "Log nr" result~getString("id") "from user" result~getString("name")" :" result~getString(log)
        /* get the next row or 0 if no more rows */
        moreRows =result~next()
    end
```

As the code box over shows the *executeQuery* method is used to retrieve data from the database. The result of the "SELECT" SQL statement is saved to the variable *result*.

The result object is of type ResultSet, and is a collection of data retrieved from the database. The ResultSet is logical divided into rows, and a pointer determines in which row the ResultSet is.
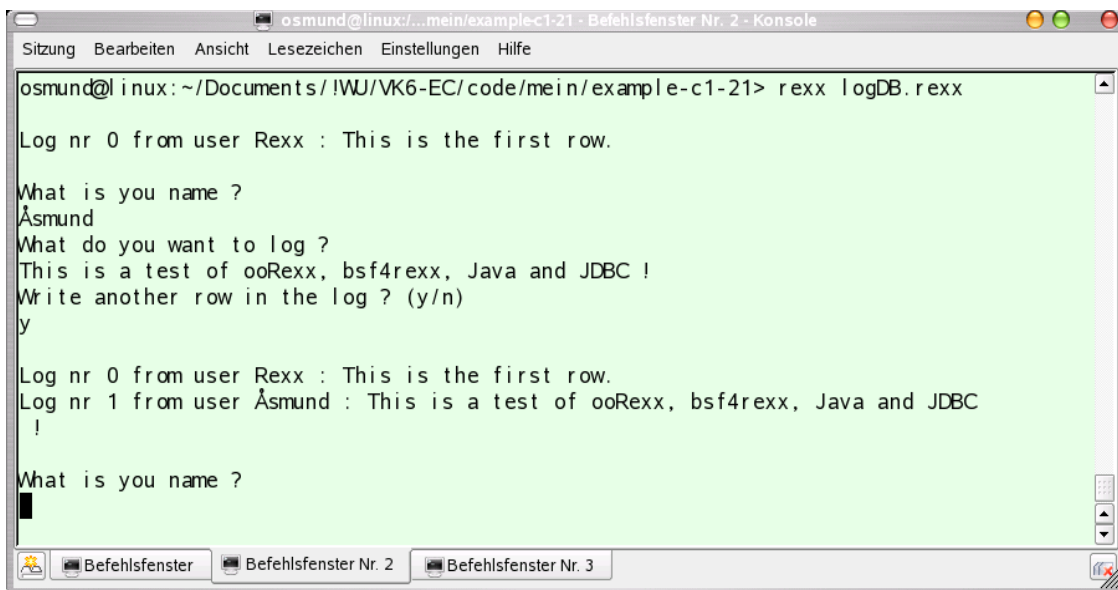
In order to set set *result* object to the first row in the ResultSet, it is necessary to call the method *next*.

When the pointer points to the first row in the DataSet the data can be retrieved using e.g. the *getString* method.

After all the data needed is retrieved from the ResultSet, the *next* method is called again.

This is repeated until the next method returns a false (0) value, meaning that all the rows in the ResultSet are processed.

The following screenshot show the out put for the user after successfully adding a row to the database.

Figure 56: Output from example C1-20

3.2 C2 – Automating OpenOffice.org with Rexx

In category 2 ooRexx and BSF4Rexx are used to control OpenOffice.org. The API of OpenOffice.org is described in more detail in chapter 2.3.

3.2.1 Example C2-1 Update a Database using BSF4Rexx

This example demonstrates how to use BSF4Rexx to update an OpenOffice.org Database by sending SQL-Statements with Rexx. This example is based on Java examples in the OpenOffice.org Developers Guide [devel05]

```
xContext = UNO.connect()  /* connect to server and retrieve the XContext object */
xMcf = xContext~getServiceManager /* get the XMultiComponentFactory */
dbc = xMcf~createInstanceWithContext("com.sun.star.sdb.DatabaseContext", xContext)
                                         /* create a DatabaseContext Object */
xn = dbc~XNameAccess /* get the NameAccess Object to the DatabaseContext Object */
xDS = xn~getbyname("teammembers") /* Get Datasource by Name */
interactionHandler=xMcf~createInstanceWithContext("com.sun.star.sdb.InteractionHandler", xContext)
                                         /* create a InteractionHandler Object */
xIntHan = interactionHandler~xInteractionHandler /* get the Interface to the InteractionHandler*/
xcompconn = xDS~XCompletedConnection /* get the Connection Object to the Datasource */
xConnection = xcompconn~connectWithCompletion(xIntHan)
                                /* create a connection using the Interactionhandler */
xStatement = XConnection~createStatement /* create a Statement Interface in the connection */
xResult = Xstatement~executeUpdate("CREATE TABLE TEAMMEMBERS (NUMMER INT PRIMARY KEY, NAME VARCHAR(50), TORE INT)")
                                         /* create a table */
xStatement~executeUpdate("INSERT INTO TEAMMEMBERS (NUMMER, NAME, TORE) VALUES (1, 'Asmund Realfsen', 7)")
                                /* insert an entry in the new table */
xStatement~executeUpdate("INSERT INTO TEAMMEMBERS (NUMMER, NAME, TORE) VALUES (2, 'Gerhard Görlich', 8)")
                                /* insert an entry in the new table */
xStatement~executeUpdate("INSERT INTO TEAMMEMBERS (NUMMER, NAME, TORE) VALUES (3, 'David Spanberger', 3)")
                                /* insert an entry in the new table */
::requires UNO.CLS    -- get UNO support
```

Figure 57: The code for example c2-1

3.2.1.1 Explanation

```
xContext = UNO.connect()
```

Using the UNO.connect method, defined in UNO.cls, to get the xContext Interface.

```
xMcf = xContext~getServiceManager
```

Get the MultiComponentFactory

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

Create the DatabaseContext Object in the above received xContext.

```
xn = dbc~XNameAccess
xDS = xn~getbyname("teammembers")
```

Get the XNameAccess Interface and get a Datasource by its name.

```
interactionHandler = xMcf~createInstanceWithContext("com.sun.star.sdb.InteractionHandler", xContext)
xIntHan = interactionHandler~xInteractionHandler
```

Get an InteractionHandler interface.

```
xcompconn = xDS~XCompletedConnection /* get the Connection Object to the Datasource */
xConnection = xcompconn~connectWithCompletion(xIntHan)
```

Connect to the Database using the InteractionHandler

```
xStatement = XConnection~createStatement
```

Create an Xstatement interface for the Connection

```
xResult=Xstatement~executeUpdate("CREATE TABLE TEAMMEMBERS (NUMMER INT PRIMARY KEY, NAME VARCHAR(50), TORE INT)")
```

Execute an update of the database by using the Xstatement Interface to send an SQL-Statement.

3.2.2 Example C2-2 – Clipboard

This example illustrates the usage of the OpenOffice.org clipboard service (*com.sun.star.datatransfer.clipboard.SystemClipboard*). The clipboard service is used for data exchange between OpenOffice.org components or between OpenOffice.org components and external applications, usually in the form of copy and paste operations [devel05, p. 419].

3.2.2.1 The Clipboard Service

The architecture of the OpenOffice.org clipboard service is strongly conforming to the Java clipboard specification [devel05, p. 419]. Since different platforms use different ways for representing clipboard data, the OpenOffice.org clipboard service uses platform independent *DataFlavor* objects as a representation for clipboard data in a certain format. *DataFlavor* objects, as defined in *com.sun.star.datatransfer.DataFlavor,* have three members devel05, p. 420]:

- MimeType – a string that describes the data conform to RfC2045 and RfC2046.

- HumanPresentableName – the human presentable name for the data format represented by the *DataFlavor* object.

- DataType – the type of data in this *DataFlavor* (the class of the data, String.class, byte[].class in Java)

```
.bsf~bsf.import("java.lang.String", "String")
xContext = UNO.connect()  -- connect to server and retrieve the XContext object
xMcf = xContext~getServiceManager  -- get the XMultiComponentFactory

/* get clipboard component*/
oClipboard = xMcf~createInstanceWithContext("com.sun.star.datatransfer.clipboard.SystemClipboard", xContext)

xClipboard = oClipboard~XClipboard

/* retrieve clipboard content */
xTransferable = xClipboard~getContents

/* get an array of "DataFlavor" objects*/
dflvArr = xTransferable~getTransferDataFlavors()

/* go through array an process the DataFlavor objects */
do df over dflvArr

 mime = df~MimeType
 hp = df~HumanPresentableName
 dt = df~DataType~toString
say "----------------------------"
/* Check if DF is supported before retrieving the data object   */
if  xTransferable~isDataFlavorSupported(df) then do
        say "MIME:" mime
        /* print only textdata in utf-8 format  */
        if pos("utf-8",mime) <> 0 then do
                data = xTransferable~getTransferData(df)
                str = .String~new(data)
                say "TEXT: " str~toString
        end
  end
end
::requires UNO.CLS    -- get UNO support
```

Figure 58: The code for example c2-2

3.2.2.2 Explanation

First, a connection to OpenOffice.org is established by using the UNO object. The result is a component context. In the next step, a service manager object is retrieved from the component context. The service manager is an object that implements the XMultiComponentFactoryFactory interface and can therefore create services (see also 3.1). By calling the *createInstanceWithContext* method with the clipboard service as parameter, a clipboard service object will be returned:

```
oClipboard = xMcf~createInstanceWithContext("com.sun.star.datatransfer.clipboard.SystemClipboard",  xContext)
```

The data in the clipboard is carried by a transferable object that implements the interface

com.sun.star.datatransfer.XTransferable. This object contains one or more *DataFlavors*, which can represent the clipboard data in different formats (e.g. in different encoded):

```
xTransferable = xClipboard~getContents
dflvArr = xTransferable~getTransferDataFlavors()
```
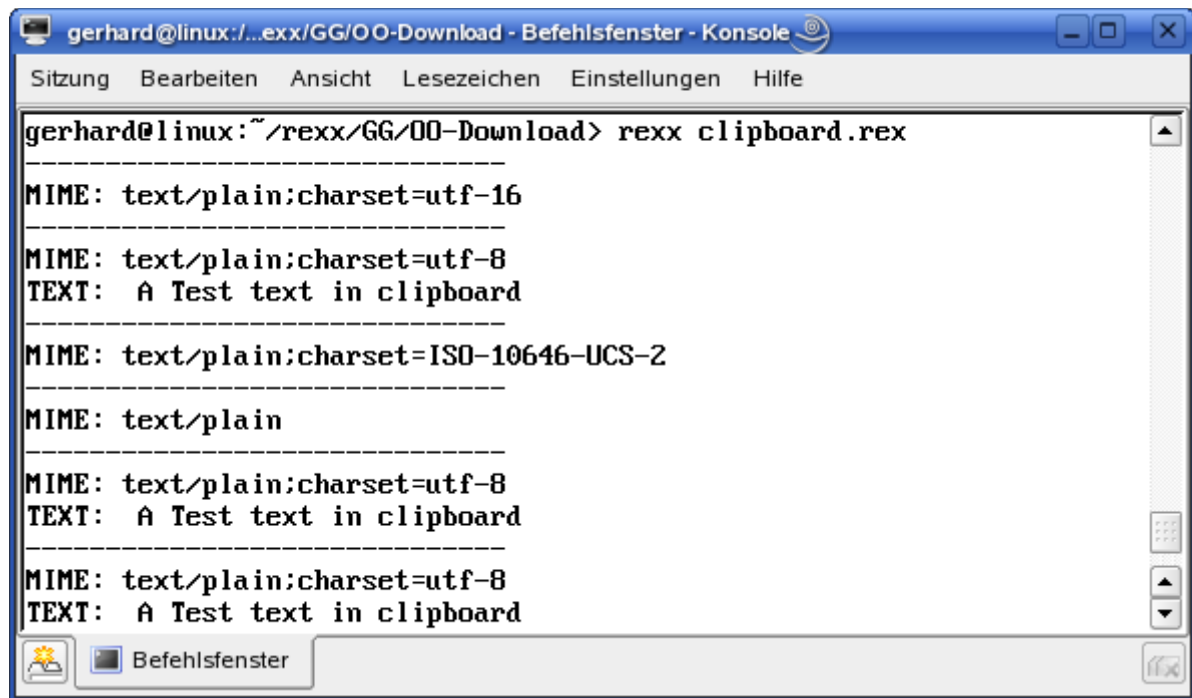
In the following loop, the format, the human presentable format and the data type for each each *DataFlavor* object are printed to the screen. If the mime type contains "utf-8", which means the data is encoded in "utf-8" format (full mime type: "text/plain;charset=utf-8"), the data will also be printed:

```
data = xTransferable~getTransferData(df)
```

The transferable object will return the data according to the matching *DataFlavor* (df).

The screenshot shows the output of this example:

Figure 59: Output from example C2-2

The clipboard data could have either type string (*String.class*) or byte array (*byte[].class*), so it can not be printed directly with *say*. This would raise an error for byte array data. The Java String class provides a constructor with a string parameter as well as one with byte array. Therefore .*String~new* can be used to assure that the data is in string format before printing.

3.2.3 Example C2-3 Print with OpenOffice.org

This example prints a existing OpenOffice.org Writer document to the standard printer.

```
/* get the xComponentLoader interface */
componentLoader = UNO.createDesktop()~XDesktop~XComponentLoader

/* start OO-Calc and open a already existing document */
fileToOpen = "file:///"directory()"/calc-print.ods"
calc=componentLoader~loadComponentFromURL(fileToOpen,"_default",0,.UNO~noProps)

/* get standard printer, then print */
printer = calc~XPrintable
printer~print(.UNO~noProps)

::requires UNO.CLS  -- load support for Open Office
```

Figure 60: The code for example C2-3

3.2.3.1 Explanation

First the *xComponentLoader* interface is retrieved. This interface is used to loads components from an URL into OpenOffice.org. [Ooo06]

Then the variable *fileToOpen* is set to point to a OpenOffice.org Calc document that already exist.

The method *loadComponentFromURL* in the *componentLoader* object is used to load the the file.

The code below shows the part that does the actually printing.

```
/* get standard printer, then print */
printer = calc~XPrintable
printer~print(.UNO~noProps)
```

First the the *XPrintable* interface is loaded from the *calc* object. Then method *print* is called from the *printer* object.

3.2.4 Example C2-4 Thesaurus

This example demonstrates how to use the OpenOffice.org thesaurus service with BSF4Rexx.

The thesaurus service is part of the OpenOffice.org Linguistic API *(com.sun.star.linguistiC2)*. This API provides also services for spell checking or hyphenation [devel05, p. 438ff]

```
.bsf~bsf.import("com.sun.star.beans.PropertyValue", "PropertyValue")
.bsf~bsf.import("com.sun.star.lang.Locale", "Locale")

xContext = UNO.connect()  /* connect to server and retrieve the XContext object */

xMcf = xContext~getServiceManager /* get the XMultiComponentFactory */

/* get the linguistics service manager */
aObj = xMcf~createInstanceWithContext("com.sun.star.linguistiC2.LinguServiceManager", xContext )

mxLinguSvcMgr = aObj~XLinguServiceManager

/* get thesaurus service */
thesaurus = mxLinguSvcMgr~getThesaurus

us = .Locale~new("en","US","")

aEmptyProps = bsf.createArray(.UNO~propertyValue, 1)     -- create Java array

do forever
        say "Enter a Word ('exit' for exit):"
        pull aWord
        if aWord = "EXIT" then leave

        xMeanings = thesaurus~queryMeanings(aWord,us,aEmptyProps)

        do meaning OVER xMeanings
                say "Meaning: " meaning~getMeaning()
                do alt over meaning~querySynonyms
                say "ALT.: " || alt
                end
        end
end

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

Figure 61: The code of example C2-4.

3.2.4.1 Explanation
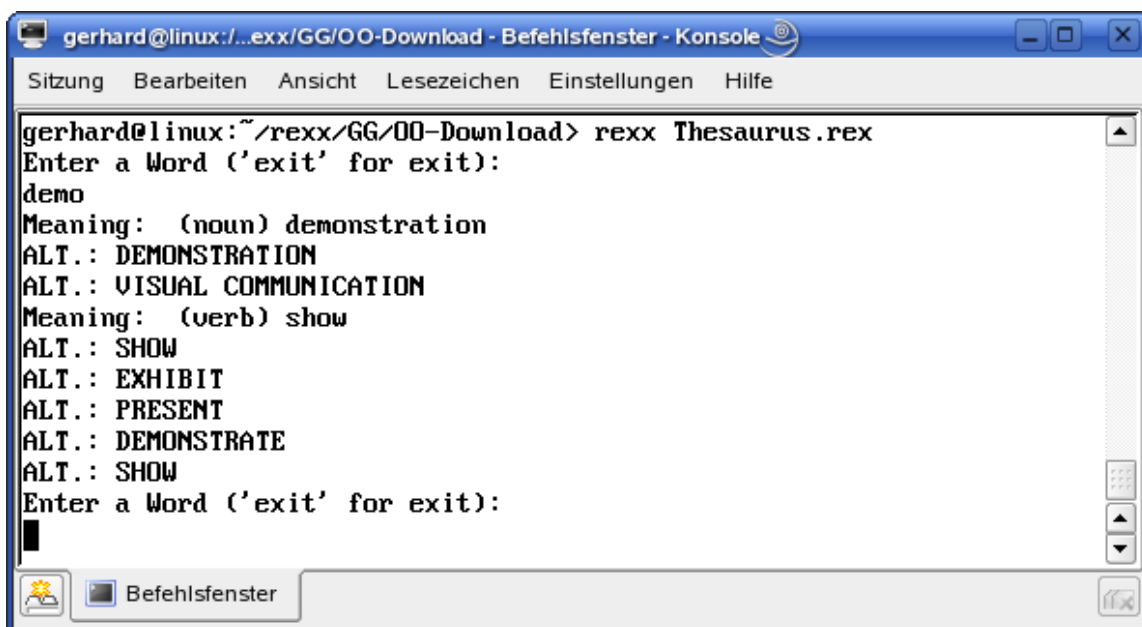
First a connection to OpenOffice.org is established and a service manager object is retrieved from the component context like in the examples before. Then a *LinguServiceManger* service is retrieved from the service manager. This object manages the interfaces for spell checker, thesaurus and hyphenation. The *getThesaurus* method returns the thesaurus service. A l*ocale* object which holds the language settings for US-English and an empty property array are prepared.

In the following loop the user can enter words, the loop is terminated with *leave* if the user enters 'exit' (independent of capitalization). The users input, the *Locale* object created before and the empty property array are the parameters for calling the thesaurus services *queryMeanings* method:

```
xMeanings = thesaurus~queryMeanings(aWord,us,aEmptyProps)
```

The method returns an array of objects implementing the *XMeanings* interface. Each *Meaning* can have multiple synonyms. These are returned by the *getSynonyms* method of the *XMeaning* object. The reason to subdivide the synonyms into different meanings is because one word can have synonyms for its different meanings that are not related. Therefore the synonyms are divided into smaller groups approximately the same definition [devel05, p. 444].

The screenshot shows the output of this example:

Figure 62: Output from example C2-4

3.2.5 Example C2-5 Cells and Charts in OO-Chart

This example opens a new OpenOffice.ord Calc document, writes some text into three cells. Then is sets the font size and the background color the three cells, and fill in numbers and formulas in 90 cells.

Next, the 90 cells containing data and formulas are used for creating a chart.

The output of the script is displayed in the screenshot below.
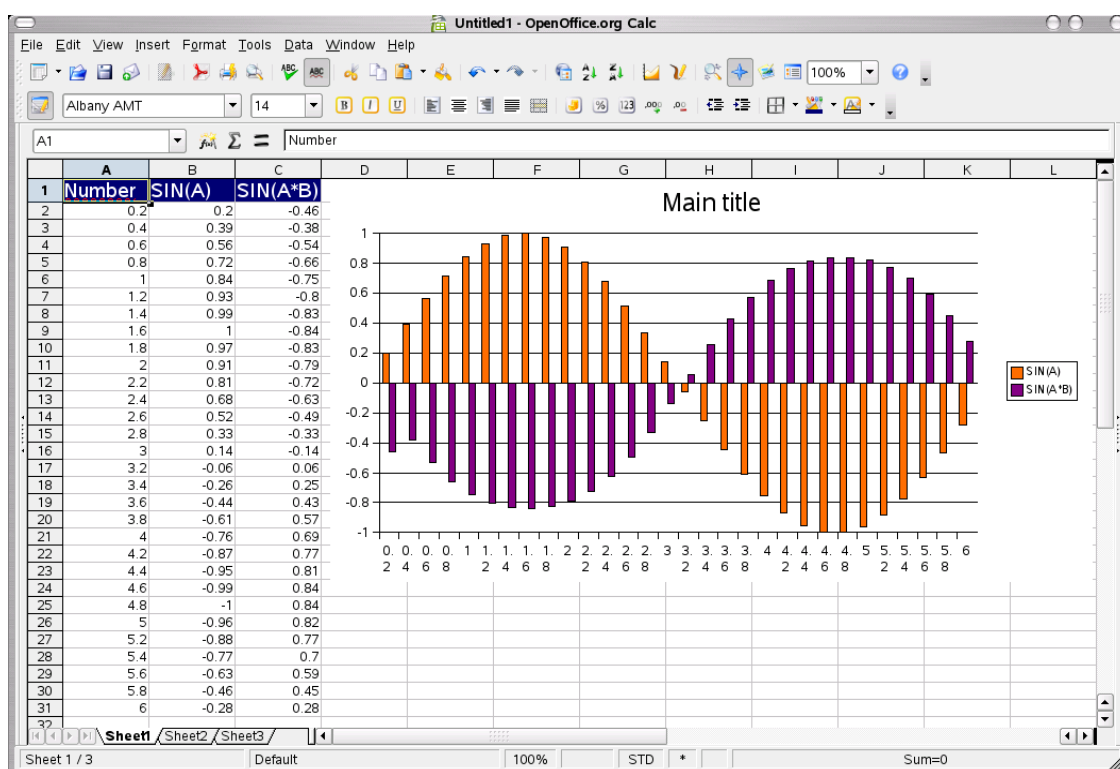
Figure 63: Screenshot of example C2-5

```
/* get the xComponentLoader interface */
componentLoader = UNO.createDesktop()~XDesktop~XComponentLoader

/* start OO-Calc with a blank document */
calcComponent = componentLoader~loadComponentFromURL("private:factory/scalc", "_blank", 0, .UNO~noProps)

/* get the first sheet in calc */
sheet=calcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* Use a procedure in uno.cls to write text into a cells */
CALL UNO.setCell sheet, 0, 0, "Number"
CALL UNO.setCell sheet, 1, 0, "SIN(A)"
CALL UNO.setCell sheet, 2, 0, "SIN(A*B)"

/* select the range a1:C1, and apply formating */
heading = sheet~getCellRangeByName("A1:C1")
heading~XPropertySet~setPropertyValue("CellBackColor", box("int", "00 00 77"x ~C2d))
heading~XPropertySet~setPropertyValue("CharHeight", box("float", "14.0"))

last_value = -1
do i=1 to 30 by 1
        /* write content to a cell using a routine from UNO.CLS */
        CALL UNO.setCell sheet, 0, i, i/5

        /* Write content to a cell without help from UNO.CLS, alternative mehtod */
        sheet~getCellByPosition(1,i)~setFormula("=SIN(A"i+1")")
        sheet~getCellByPosition(2,i)~setFormula("=SIN(A"i+30"-B"i+1")")
end

/* create dimensions for chart */
chartRec = .bsf~new("com.sun.star.awt.Rectangle")
chartRec~X = 7000
chartRec~Y = 0
chartRec~Width = 20000
chartRec~Height = 10000

/* select cell range for chart */
chartCellRange = sheet~getCellRangeByName("A1:C31")~xCellRangeAddressable~getRangeAddress

/* Create an array with data to insert into the chart */
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
chartAddressArray = bsf.createArray(.UNO~CellRangeAddress, 1)
chartAddressArray[1] = chartCellRange

/* create and show chart */
chart = sheet~xTableChartsSupplier~getCharts
chart~addNewByName("myChart", chartRec, chartAddressArray, .true, .true)

::requires UNO.CLS  -- load support for Open Office
```

Figure 64: The code for example C2-5

3.2.5.1 Explanation

First the xComponentLoader interface is used to start the OpenOffice.org Calc component.

Then the variable *sheet* is referenced to the first sheet in the document. The first sheet of the document has the index 0.

Next, the code in the box bellow is executed.

```
/* Use a procedure in uno.cls to write text into a cells */
CALL UNO.setCell sheet, 0, 0, "Number"
CALL UNO.setCell sheet, 1, 0, "SIN(A)"
CALL UNO.setCell sheet, 2, 0, "SIN(A*B)"
```

All UNO methods are methods provided from the file UNO.CLS. UNO.CLS contains methods that makes it simpler to automate OpenOffice.org from ooRexx. UNO.CLS is a part of the BSF4Rexx package.

The *setCell* method of UNO.CLS is method that writes values or formulas into cells. The first argument is a reference to which sheet to use. The next arguments are the adress of the cell, and the value or formula to insert.

The alternative to the method setCell from the UNO.CLS file, is to first get the cell position, then use the *setFormula* method.

```
sheet~getCellByPosition(1,i)~setFormula("=SIN(A"i+1")")
```

The alternative version as displayed above, is used later in the script.

After the headings are written the formatting is applied. The *getCellRangeByName* method select a area in the spreadsheet. In this case the area A1 to C1 is selected.

```
/* select the range a1:C1, and apply formating */
heading = sheet~getCellRangeByName("A1:C1")
heading~XPropertySet~setPropertyValue("CellBackColor", box("int", "00 00 77"x ~C2d))
heading~XPropertySet~setPropertyValue("CharHeight", box("float", "14.0"))
```

In order apply formating on the selected area, the interface *XpropertySet* has to be used. With this interface, information on the selected area can be retrieved and properties can be set. With the *setPropertyValue* method the background color and the font size are specified.

The second argument of the *setPropertyValue* method is the value to set the property to use. The *box* function is specified from BSF4Rexx and makes sure that the specified value is returned. The *C2d* function on the line that sets the background color, is a ooRexx function that converts the hexadecimal value into a decimal value. This again returned as a integer value because of the *box* function in BSF4Rexx.

After the formating of the first row is set, the script write values and formulas into the area A2 to C31 in the sheet. This is done using the methods presented above.

Then the dimension of the chart is created using an object from the *com.sun.star.awt.Rectangle* class.

Next the variable *chartCellRange* is set to the range A2 to C31. This is the data to use when creating the chart.

A chart can contain data from several ranges of cells. To archive this functionality an array of cell ranges has to be created. This example use only one cell range, but with just small changes in the code, several cell ranges could be used.

After the cell range is inserted into an array, the chart it self is created.

3.3 C3 – Combining Java APIs and OpenOffice.org with ooRexx

In this section the usage of Java APIs for enhancing the abilities of OpenOffice.org is shown.

3.3.1 Example C3-1 Inserting Charts in OpenOffice.org Draw

This example demonstrates how to use BSF4Rexx to create a similar Chart with JfreeChart and OpenOffice.org Calc based on data from a database in OpenOffice.org Base. The time it takes to create each of the charts gets measured. After creating these charts, they are inserted into a Draw document together with a box containing the measured time. JfreeChart has to be installed to run this example.

This example is contains parts of Java examples in the OpenOffice.org Developers Guide ([devel05]) and parts of OpenOffice.org Automation nutshell examples in [Aha05].
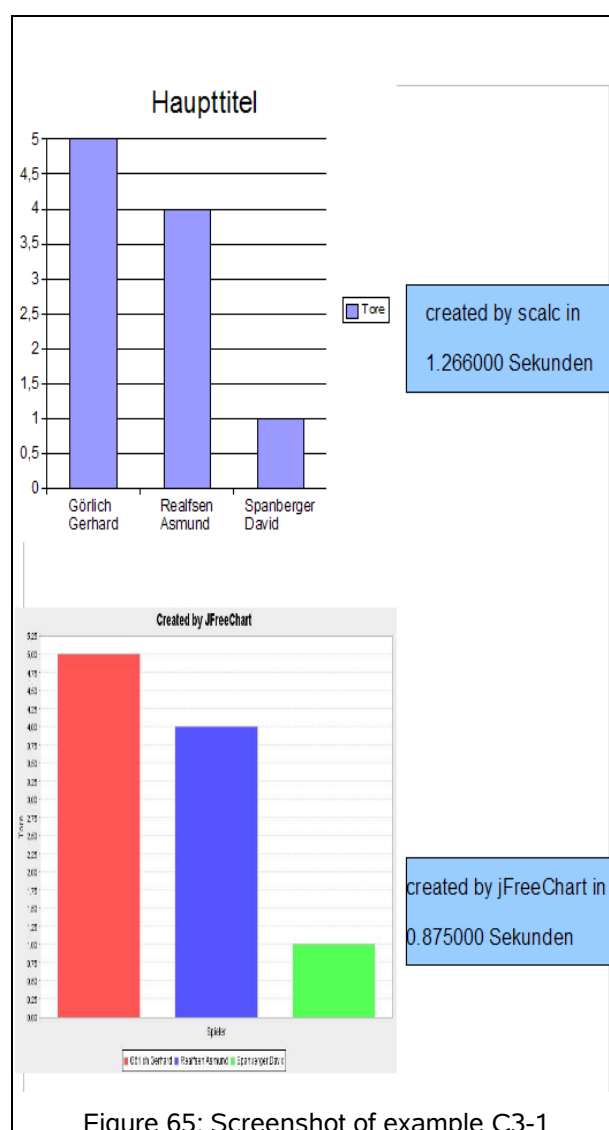
Figure 65: Screenshot of example C3-1

```
/*------------------------- create a New Draw Document -------------------------*/
oDesktop         = UNO.createDesktop() /* get the desktop Object */
xComponentLoader = oDesktop~XDesktop~XComponentLoader
                               /* get componentLoader interface of the Desktop Object */
url = "private:factory/sdraw" /* url to the blank *.sxw - file */
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0,.UNO~noProps)
                                         /* open the url with no Properties */
xDMsf = xDrawComponent~XMultiServiceFactory /* Get the MultiServiceFactory of the draw Object */
xDrawPage=xDrawComponent~XDrawPagesSupplier~getDrawPages~getByIndex(0)~XDrawPage
                                         /* get the first Page in the Draw Document */
/*-----------------------------------------------------------------------------*/

/*------------------------- get the Database Connection -------------------------*/
xContext = UNO.connect()  /* connect to server and retrieve the XContext object */
XMcf = xContext~getServiceManager /* get the XMultiComponentFactory */
oRowSet = xMcf~createInstanceWithContext("com.sun.star.sdbc.RowSet", xContext)
                                         /* create RowSet Object */
xRowSet = oRowSet~XRowSet /* get it's interface */
xProp = xRowSet~XPropertySet  /* create a Properties Object for the connection to the database */
xProp~setPropertyValue("DataSourceName", "teammembers")
                             /* the Name of database we want to connect to */
xProp~setPropertyValue("CommandType", box("int", bsf.getStaticValue("com.sun.star.sdb.CommandType", "COMMAND")))
                                         /* set the Command Type*/
xProp~setPropertyValue("Command", "SELECT Nachname, Vorname, Tore FROM table_names ORDER BY ID")
/* The Command we want to execute */
xRowSet~execute /* execute the command */
xRow = oRowSet~XRow  /* get column access to the row object */
/*-----------------------------------------------------------------------------*/

/*------------------------- create Graph using JFreeChart -------------------------*/
nullsetzen=time(R) /* Reset Time to get the time it takes to create the Graph */
.bsf~bsf.import("org.jfree.chart.ChartFactory","JChartFactory")
                                         /* import the JFreeChart Chartfactory Class */
.bsf~bsf.import("org.jfree.chart.ChartUtilities","JChartUtilities")
                                         /* import the JFreeChart Chartutilities Class */
.bsf~bsf.import("org.jfree.chart.JFreeChart","JFreeChart")
                                         /* import the JFreeChart JFreeChart Class */
.bsf~bsf.import("org.jfree.data.category.DefaultCategoryDataset","DefaultCategoryDataset")
                                 /* import the JFreeChart DefaultCategoryDataset Class */
.bsf~bsf.import("java.io.File","JFile") /* import the Java File I/O Class */
.bsf~bsf.import("org.jfree.chart.plot.PlotOrientation","PlotOrientation")
                                 /* import the JFreeChart PlotOrientation Class */
dataset=.DefaultCategoryDataset~new()   /* create a new DefaultCategoryDataset Object */
DO WHILE xRowSet~next > 0 /* create loop, which runs until it has reached the end of the RowSet */
       dataset~addValue(xRow~getString(3),xRow~getString(1) || " " || xRow~getString(2),"")
                                         /* add the Values to the Dataset */
END
chart=.JChartFactory~createBarChart("Created by JfreeChart","Spieler","Tore",dataset,.PlotOrientation~VERTICAL,.true,.true,.false)
                                         /* create the Bar Chart */
fileName="teammembers.jpg"  /* set the output file name */
chartFile=.JFile~new(fileName) /* create a new Java File Object */
.JChartUtilities~saveChartAsJPEG(chartFile, chart, 800,500) /* save the file as JPG */
/* Insert the JPG in the Draw Document */
oGraph = xDMsf~createInstance("com.sun.star.drawing.GraphicObjectShape")
                                         /* create a new GraphicObjectShape Object */
xGraph = oGraph~xShape /* get the Interface to the GraphicObjectShape Object */
size = .bsf~new("com.sun.star.awt.Size") /* create a new Size Object */
point = .bsf~new("com.sun.star.awt.Point") /* create a new Point Object */
size~Height = 13000 /* set height */
size~Width = 13000 /* set width */
point~x = 200 /* set coordinates on the x axis, where to insertthe Object */
point~y= 15000 /* set coordinates on the y axis, where to insertthe Object */
xGraph~setSize(size) /* apply the Size Object to the GraphicObjectShape Object */
xGraph~setPosition(point)  /* apply the Point Object to the GraphicObjectShape Object */
xPropertySet=xGraph~xPropertySet /* get PropertySet Interface to the GraphicObjectShape Object */
xPropertySet~setPropertyValue("GraphicURL", makeURL("teammembers.jpg"))
                                         /* set the Filename of the File to insert */
xDrawPage~add(xGraph) /* add Graph to the draw Page */
jfreetime=time(E) /* get Time it takes to create the graph with JFreeChart */

oleShapeProps~setPropertyValue("CLSID", msChartClassID)
                                         /* set CLSID Value to the Propertyset */
model = oleShapeProps~getPropertyValue("Model") /* get the Model Value */
xChartDocument = model~xChartDocument /* get the XChartdocument Interface */
xChartDocument~attachdata(xDiagram) /* attach data to the xChartDocument */
scalctime=time(E) /* get Time it takes to create the graph with scalc */
/*-----------------------------------------------------------------------------*/
```

Figure 66: The code of example C3-1, part 1

```
/*------------------------ add Rectangles with text to the Document ------------------------*/
/* Rectangle Nr. 1 */
shapeRectangle = -
        xDMsf~createInstance("com.sun.star.drawing.RectangleShape")
                                                /* create new rectangleShape Object */
xShapeRectangle = shapeRectangle~XShape              /* Get the Xshape Interface */
shapeX = 13500
shapeY = 6000
xShapeRectangle~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))
                                                /* set the position of the Rectangle */
shapeWidth  = 7000
shapeHeight = 3000
xShapeRectangle~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
                                                /* set the size of the Rectangle */
xDrawPage~add(xShapeRectangle)                       /* add the Rectangle to the DrawPage */
xText = xShapeRectangle~XText                        /* Get the Xtext Interface */
xTextCursor = xText~createTextCursor                 /* create the TextCursor */
xTextCursor~gotoEnd(.false)                          /* set cursor to the end */
xTextRange = xTextCursor~XTextRange                  /* get the XTextRange Interface */
xTextRange~setString("created by scalc in " || "0a0a"x || scalctime || " Sekunden")
                                                /* add Text */

/* Rectangle Nr. 2 */
shapeRectangle =  xDMsf~createInstance("com.sun.star.drawing.RectangleShape")
                                      /* create new rectangleShape Object */
xShapeRectangle = shapeRectangle~XShape              /* Get the Xshape Interface */
shapeX = 13500
shapeY = 22000
xShapeRectangle~setPosition(.bsf~new("com.sun.star.awt.Point", shapeX, shapeY))
                                                /* set the position of the Rectangle */
shapeWidth  = 7000
shapeHeight = 3000
xShapeRectangle~setSize(.bsf~new("com.sun.star.awt.Size", shapeWidth, shapeHeight))
/* set the size of the Rectangle */
xDrawPage~add(xShapeRectangle)                       /* add the Rectangle to the DrawPage */
xText = xShapeRectangle~XText                        /* Get the Xtext Interface */
xTextCursor = xText~createTextCursor                 /* create the TextCursor */
xTextCursor~gotoEnd(.false)                          /* set cursor to the end */
xTextRange = xTextCursor~XTextRange                  /* get the XTextRange Interface */
xTextRange~setString("created by jFreeChart in " || "0a0a"x || jfreetime || " Sekunden")
                                                /* add Text */

::requires UNO.CLS /* load UNO support for OpenOffice.org */
::routine makeUrl    /* operating system independent */
return  ConvertToURL(stream(arg(1), "c", "query exists"))
```

Figure 67: The code of example C3-1, part 3

3.3.1.1 Explanation

```
/*-------------------------- create a New Draw Document --------------------------*/
oDesktop         = UNO.createDesktop() /* get the desktop Object */
xComponentLoader = oDesktop~XDesktop~XComponentLoader
                                /* get componentLoader interface of the Desktop Object */
url = "private:factory/sdraw" /* url to the blank *.sxw - file */
xDrawComponent = xComponentLoader~loadComponentFromURL(url, "_blank", 0,.UNO~noProps)
                                                /* open the url with no Properties */
```

These lines open a new empty Draw document.

```
xDMsf = xDrawComponent~XMultiServiceFactory /* Get the MultiServiceFactory of the draw Object */
xDrawPage=xDrawComponent~XDrawPagesSupplier~getDrawPages~getByIndex(0)~XDrawPage
```

Get the MultiComponentFactory and assign the "XdrawPage" Interface of the first Page of the document to the variable "xDrawPage"

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

Create the DatabaseContext Object in the above received xContext.

```
nullsetzen=time(R)
```

Reset the time counter for measuring the time.

```
dataset=.DefaultCategoryDataset~new()    /* create a new DefaultCategoryDataset Object */
DO WHILE xRowSet~next > 0 /* create loop, which runs until it has reached the end of the RowSet */
        dataset~addValue(xRow~getString(3),xRow~getString(1) || " " || xRow~getString(2),"")
                                         /* add the Values to the Dataset */
END
```

Create a new Dataset and assign the values from the database to it.

```
/* create the Bar Chart */
chart=.JChartFactory~createBarChart-
("Created by JfreeChart","Spieler","Tore",dataset,.PlotOrientation~VERTICAL,.true,.true,.false)
fileName="teammembers.jpg"   /* set the output file name */
chartFile=.JFile~new(fileName) /* create a new Java File Object */
.JChartUtilities~saveChartAsJPEG(chartFile, chart, 800,500) /* save the file as JPG */
```

Create a Barchart and save it as an Jpg-Image.

```
/* create a new GraphicObjectShape Object */
oGraph = xDMsf~createInstance("com.sun.star.drawing.GraphicObjectShape")
xGraph = oGraph~xShape /* get the Interface to the GraphicObjectShape Object */
```

Create a new GraphicObjectShape and get its Xshape Interface.

```
size = .bsf~new("com.sun.star.awt.Size") /* create a new Size Object */
point = .bsf~new("com.sun.star.awt.Point") /* create a new Point Object */
size~Height = 13000 /* set height */
size~Width = 13000 /* set width */
point~x = 200 /* set coordinates on the x axis, where to insertthe Object */
point~y= 15000 /* set coordinates on the y axis, where to insertthe Object */
xGraph~setSize(size) /* apply the Size Object to the GraphicObjectShape Object */
xGraph~setPosition(point)  /* apply the Point Object to the GraphicObjectShape Object */
```

Set the size and the position of the GraphicObjectShape.

```
xPropertySet=xGraph~xPropertySet /* get PropertySet Interface to the GraphicObjectShape Object */
/* set the Filename of the File to insert */
xPropertySet~setPropertyValue("GraphicURL", makeURL("teammembers.jpg"))
```

Define which image has to be inserted in the GraphicObjectShape.

```
xDrawPage~add(xGraph) /* add Graph to the draw Page */
jfreetime=time(E) /* get Time it takes to create the graph with JFreeChart */
```

Add the GraphicObjectShape to the Draw document. Get the time since the last time reset

to know how long it took to create this chart.

```
props[1] = .UNO~PropertyValue~new /* make the array element to a property Value */
props[1]~Name = "Hidden" /* set the property to open the file hidden */
props[1]~Value = box("boolean", .true) /* set the property Value .true */
```

Open the Calc component hidden. This means it is invisible to the user.

```
DO WHILE xRowSet~next > 0  /* create loop, which runs until it has reached the end of the RowSet */
        CALL UNO.setCell xSheet, 0, i, xRow~getString(1) || " " || xRow~getString(2)
                                                /* put Data in the first column */
        CALL UNO.setCell xSheet, 1, i, xRow~getString(3)  /* put Data in the second column */
        i=i+1
END
```

Insert the values from the database into the cells of the Calc Document.

```
myRange=xSheet~XCellRange~getCellRangeByName("A1:B" || i)
                                        /* get CellRange of Cells which are used for the graph */
myAddr = myRange~XCellRangeAddressable~getRangeAddress /* get the Address of the CellRange */
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress" /* create the CellRangeAddress for the Chart */
oAddr = bsf.createArray(.UNO~CellRangeAddress, 1) /* create Java array */
oAddr[1] = myAddr
```

Select the Cell Range, which should be used for the chart.

```
oRect = .bsf~new("com.sun.star.awt.Rectangle")  /* create a rectangle for the chart */
oRect~X      = 0
oRect~Y      = 0
oRect~Width  = 18000
oRect~Height = 8000
```

Create a rectangle, in which the chart will be inserted.

```
xTableCharts = xSheet~XTableChartsSupplier~getCharts /* get the Sheet's ChartsSupplier */
xTableCharts~addNewByName("Tore", oRect, oAddr, .true, .true) /* add the new Chart */
```

Create the new chart.

```
ole2shape = xDrawFactory~createInstance("com.sun.star.drawing.OLE2Shape")~xShape
                                        /* create an OLE3Shape Object and its interface */
xDrawPage~add(ole2shape) /* add the shape to the DrawPage */
```

Create a new OLE2Shape and add it to the draw Page.

```
msChartClassID = "12dcae26-281f-416f-a234-C3086127382e"
oleShapeProps = ole2shape~xPropertySet /* get Propertyset of the ole2shape */
oleShapeProps~setPropertyValue("CLSID", msChartClassID) /* set CLSID Value to the Propertyset */
model = oleShapeProps~getPropertyValue("Model") /* get the Model Value */
xChartDocument = model~xChartDocument /* get the XChartdocument Interface */
xChartDocument~attachdata(xDiagram) /* attach data to the xChartDocument */
```

Attach the diagram into the OLE2Shape.

```
xText = xShapeRectangle~XText /* Get the Xtext Interface */
xTextCursor = xText~createTextCursor /* create the TextCursor */
xTextCursor~gotoEnd(.false) /* set cursor to the end */
xTextRange = xTextCursor~XTextRange /* get the XTextRange Interface */
xTextRange~setString("created by scalc in " || "0a0a"x || scalctime || " Sekunden") /* add Text */
```

Write the measured time into the rectangles next to the graphs.

This example is big and complex. Therefore it is difficult to get the overview over the whole program. But if you understand the examples above it, there is no real problem.

3.3.2 Example C3-2 Regexp and Charts

This example combines the knowledge from example C1–2 (regular expressions) and C2–6 (cells and charts).

The script downloads the pages from finance.yahoo.com for a list of stock symbols. Then these pages are parsed via the Java regular expression classes. The values are inserted into a Calc sheet and a chart is generated.

```
/* import Java classes */
.bsf~bsf.import("java.io.InputStream", "InputStream")
.bsf~bsf.import("java.net.URL", "URL")
.bsf~bsf.import("java.io.DataInputStream", "DataInputStream")
.bsf~bsf.import("java.io.BufferedInputStream", "BufferedInputStream")

.bsf~bsf.import("java.util.regex.Pattern", "Pattern")
.bsf~bsf.import("java.util.regex.Matcher", "Matcher")


/********* Start Main routine ****************************/

/* get the xComponentLoader interface */
componentLoader = UNO.createDesktop()~XDesktop~XComponentLoader

/* start OO-Calc with a blank document */
calcComponent = componentLoader~loadComponentFromURL("private:factory/scalc", "_blank", 0, .UNO~noProps)

/* get the first sheet in calc */
sheet = calcComponent~XSpreadSheetDocument~getSheets~XIndexAccess~getByIndex(0)~XSpreadSheet

/* Use a procedure in uno.cls to write text into a cells */
CALL UNO.setCell sheet, 0, 0, "Stock"         -- this is the heading of col one
CALL UNO.setCell sheet, 1, 0, "Last Trade"    -- this is the heading of col two

/* select the range a1:c1, and apply formating */
heading = sheet~getCellRangeByName("A1:C1")
heading~XPropertySet~setPropertyValue("CellBackColor", box("int", "00 00 77"x ~c2d))
heading~XPropertySet~setPropertyValue("CharHeight", box("float", "14.0"))

row = 1

/* Some stock symbols we want to query  */
stocks = bsf.createArray("String.class", 6)
stocks[1] = ADBE
stocks[2] = GOOG
stocks[3] = YHOO
stocks[4] = DTE.DE
stocks[5] = IDS.DE
stocks[6] = TOI.DE

/* retreive stocks from yahoo stock service, may take a bit time */
do stock over stocks
        /* Call sub_routine get_stock_info */
        call get_stock_info (stock)
        last_trade = result
        /* write result to a cell using a routine from UNO.CLS */
        CALL UNO.setCell sheet, 0, row, stock
        CALL UNO.setCell sheet, 1, row, last_trade

        row = row +1
end


/* create dimensions for chart */
chartRec = .bsf~new("com.sun.star.awt.Rectangle")
chartRec~X = 7000
chartRec~Y = 0
chartRec~Width = 20000
chartRec~Height = 10000

/* select cell range for chart */
chartCellRange = sheet~getCellRangeByName("A1:B7")~xCellRangeAddressable~getRangeAddress

/* Create an array with data to insert into the chart */
CALL UNO.loadClass "com.sun.star.table.CellRangeAddress"
chartAddressArray = bsf.createArray(.UNO~CellRangeAddress, 1)
chartAddressArray[1] = chartCellRange

/* create and show chart */
chart = sheet~xTableChartsSupplier~getCharts
chart~addNewByName("Stocks", chartRec, chartAddressArray, .true, .true)

exit
```

```
/*************** Begin function get_stock_info ************************/

get_stock_info:
parse arg symbol

/* create an url object */
url = .URL~new("http://finance.yahoo.com/q/bc?s="symbol"&t=3m")

/* open url as stream */
is = url~openStream()

/* make stream a data input stream */
dis = .DataInputStream~new(.BufferedInputStream~new(is))

/* read the webpage line by line */
s = dis~readLine()
st = ""
do while s <> .nil
st = st s
/* say s */
s = dis~readLine()

end

say str

pattern = .Pattern~compile("Last Trade:</td>[^/]*");
matcher = pattern~matcher(st)

found = 0

do while matcher~find() <> 0
        group =  matcher~group()  /* " starting at index "  matcher~start()   " and ending at index
"matcher~end() "." */

        p2 = .Pattern~compile("<b>(.*)<")
        m2 = p2~matcher(group)
        m2~find()
        say "last Trade: " m2~group()
        last_trade = m2~group()
            found = 1
end

        if found = 0 then say "No match found."
is~close()
nrend = last_trade~length() - 5
say nrend
last_trade = last_trade~substr(4, nrend)

return last_trade

::requires UNO.CLS  -- load support for Open Office
```

3.3.2.1 Explanation

This example is divided into two parts: the main routine and the *getStockinfo* function.

The main routine starts with importing the necessary classes, connecting to OpenOffice.org and retrieving the service manager from the component context. The next steps are like in example C2–6: A new Calc document is created, the headlines are written into cells and the cells are being formatted. Then an array is defined which contains the stock symbols to be processed. In a loop, the last trade value for each symbol is received from the *getStockinfo* function and then, as the stock symbol itself, written into a new cell in a new row. The results are two columns filled with "stock symbol" - "value" pairs. Then a chart is created, like in example C2-6, for the cell range A1:B7.

The *getStockinfo* function determines the last trade value for a stock symbol. It starts by downloading a page from finance.yahoo.com with the passed stock symbol as a part of the

URL. For downloading, the page is simply opened as a Java *BufferedStream* and then read line by line:

The resulting string is used as input for the *Pattern* objects *matcher* method, as in example C1–2. The resulting string still contains more characters than only the last trade value, so another regular expression pattern is applied. Finally the *substr* method cuts of the last dispensable characters and the plain value is returned.
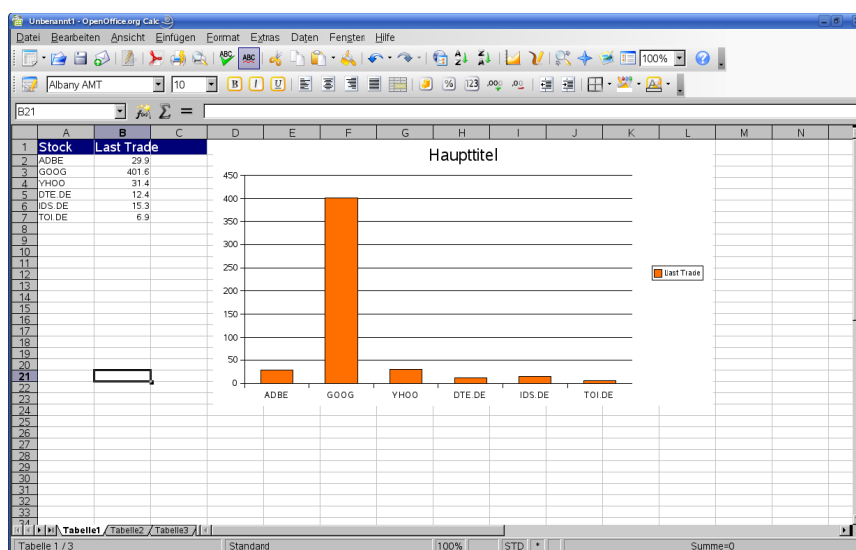

Figure 68: Screenshot of example C3-2

3.3.3 Example C3-3 FreeTTS and OpenOffice.org

The following script use the text to speech engine FreeTTS presented in example C1-12 to let the computer speak the text of an OpenOffice.org Writer document.

In order to run this example the FreeTTS library need to be downloaded and installed. FreeTTS can be downloaded from this address: http://freetts.sourceforge.net/docs/index.php

A detailed description of how to install external Java libraries can be found in the beginning of this paper.

This script is designed to run within OpenOffice as a macro.

The best way to archive this is to copy the script into a new macro in OpenOffice.org.

A new macro is created from the *Tools>>Macros>>Organize Macros>>ooRexx* menu. First a new library has to be created, and then a new macro can be placed inside of it.

The macro is executed from the *Tools>>Macros>>Run Macro* menu.

```
/* import classes */
.bsf~bsf.import("com.sun.speech.freetts.Voice","JVoice")
.bsf~bsf.import("com.sun.speech.freetts.VoiceManager","JVoiceManager")
.bsf~bsf.import("com.sun.speech.freetts.audio.JavaClipAudioPlayer","JavaClipAudioPlayer")

/* get document and text in document */
doc=uno.getScriptContext()~getDocument
text=doc~XTextDocument~getText

/* create object using static method */
voiceManager = .JVoiceManager~getInstance();

/* get an instance of the voice engine */
voice = voiceManager~getVoice("kevin16")

/* activate the voice engine */
voice~allocate()

voice~speak(text~getString())

/* unbind the voice engine */
voice~deallocate()

::requires UNO.CLS   -- load UNO support for OpenOffice.org
```

Figure 69: The code for example C3-3

3.3.3.1 Explanation

FreeTTS is explained in detail in example C1-12, and will not be explained in this example.

Running a script inside OpenOffice.org as a macro is simpler than trying to control OpenOffice.org from outside.

This example only contains three OpenOffice.org specific lines of code.

First a reference to the document and the text in the document is loaded. The *text* object is the object that holds the text and must not be confused with the text content it self. To get the actual text, the method *getString* on the *text* object has to be used. The result is a string with all the text in the document, regardless of the number of pages. Hence this string can be quite long.

The resulting string is passed into FreeTTS for audio output as displayed in the box bellow.

```
voice~speak(text~getString())
```

4 Conclusion and Future Prospects

The BSF4Rexx is a very useful technology to enhance the functionality of ooRexx. With BSF4Rexx it is possible to write ooRexx scripts which can use any Java library. This is particularly useful in cases where only Java libraries provide the required functionality.

In the first part of this work the system architecture and a guide of how to install all the components are presented. The second part shows how to use these components to use Java in ooRexx. This is demonstrated with the help of small nutshell examples, which are developed by the authors of this paper. Additionally this work includes examples of how to automate OpenOffice.org by using the UNO Component via Java and BSF4Rexx.

During development of this paper, some problems were encountered. This was special the case of the part working with OpenOffice.org. The OpenOffice.org DevelopersGuide [devel05] explains how to automate OpenOffice.org with Java. But there are several errors in this guide. It is very difficult to develop programs, based on a guide with errors. The authors had to use the mailing lists to ask the communities, why things that do not work that way they are mentioned in the Developers Guide.

As the examples in this paper shows, BSF4Rexx can be applied to a wide range of user scenarios. However, which scenarios that are really usefully in the real world, is something time will show. In some cases it might be better to program directly in Java. In other cases it might be better to solve the problem using only ooRexx. But there are certainly a lot of scenarios where the connection between ooRexx and Java makes sense.

The problem with BSF4Rexx is that it until now there did not exist a lot of examples or documentation of how apply it in a real world context. This is specially the case for BSF4Rexx in relation to OpenOffice.org. This paper covers a lot of possible scenarios, but none of them are covered really deep. Hence, there is still a lot of room for future research and development of possible user scenarios and use of BSF4Rexx in a real world context.

The authors of this paper hope that the examples presented in this work can serve as a guide for people learning BSF4Rexx, and that this work will contribute to a wider acceptance and use of this technology.

5 References

[Aha05]   Ahammer Andreas, OpenOffice.org Automation: Object Model, Scripting
             Languages, "Nutshell"-Examples, 2005, Wirtschaftsuniversität Wien
             (Vienna University of Economics and Business Administration), Austria;
             URL (2006-06-20):

             http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2005/200511_OOo-
             Ahammer/200511_OOoAutomation.pdf

[AJP06]   Apache Jakarta Project homepage. URL (2006-06-22):
             http://jakarta.apache.org

[AJP06a]   Apache Jakarta Project homepage: BSF Documentation, URL (2006-06-
             22): http://jakarta.apache.org/bsf/manual.html

[Aug05]   Augustin Walter, Examples for Open Office Automation with Scripting
             Languages, 2005, Wirtschaftsuniversität Wien (Vienna University of
             Economics and Business Administration), Austria; URL (2006-06-20):
             http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2005/200501_OOo-
             Agustin/200501_BSF-Examples_Augustin.pdf

[Burger05]   Burger Martin, OpenOffice.org Automation with Object Rexx, 2005,
             Wirtschaftsuniversität Wien (Vienna University of Economics and Business
             Administration), Austria; URL (2006-06-20):
             http://wi.wu-
             wien.ac.at/rgf/diplomarbeiten/BakkStuff/2006/200605_Burger/Bakk_Arbeit
             _Burger20060519.pdf

[devel05]   OpenOffice.org 2.0 Developers Guide, Sun Microsystems, May 2005

[Flat06]   Flatscher Rony G., The Vienna Version of BSF4Rexx, 2006, Presentation
             at the 2006 International Rexx Symposium, USA; URL (2006-06-20):
             http://wi.wu-wien.ac.at/rgf/Rexx/orx17/2006_orx17_BSF_ViennaEd.pdf

[Hsq06]   HSQLDB. 2006-04-16. HSQLDB. URL (2006-06-21): http://hsqldb.org/

[J3D00]   Getting Started With Java 3D, Dennis J. Bouvier, Sun Microsystems 1999-
             2000, URL (2006-05-10):

http://java.sun.com/developer/onlineTraining/java3d/j3d_tutorial_ch1.pdf

[JFree06]   FreeChart Homepage, URL (2006-06-10):

http://www.jfree.org/jfreechart/

[JLayer06]   JLayer Homepage, URL (2006-06-10)

http://www.javazoom.net/javalayer/javalayer.html

[Kru05]   Krüger Guido, Handbuch der Java-Programmierung, 2005, 4th edition Addison Wesely. Online Version (URL 2006-06-20):

http://www.linguistik.uni-erlangen.de/~arviktor/k100304.html

[LGPL06]   GNU Lesser General Public License, URL (2006-06-10):

http://www.gnu.org/licenses/lgpl.html

[OASIS06]   OASIS Open Document Format, URL (2006-06-13):

http://www.oasis-open.org/committees/office/charter.php

[OO06]   The OpenOffice.org project homepage, URL (2006-06-11):

http://www.openoffice.org

[OOo06]   Interface XcomponentLoader. OpenOffice.org API. URL (2006-06-22):

http://api.openoffice.org/docs/common/ref/com/sun/star/frame/XComponentLoader.html

[ooR05]   About Open Object Rexx. 2005-04-27. Rexx Language Association. URL (2006-06-22): http://www.ooRexx.org/

[portOO06]   Portable OpenOffice.org, URL (2006-06-13):

http://portableapps.com/apps/office/suites/portable_openoffice

[reflect06]   The Java Tutorial, Trail: Reflection, by Dale Green, URL (2006-06-14):

http://java.sun.com/docs/books/tutorial/reflect/index.html

[regexp06]   Java 1.4.2 API Documentation, java.util.regexp package, URL (2006-06-10):

http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html

[Star06]   The StarOffice homepage, URL (2006-06-13):

http://www.sun.com/software/star/staroffice/index.jsp

[Sun04]   JavaTM Cryptography Architecture - API Specification & Reference. 2004-07-24. Sun Microsystems, Inc. URL (2006-06-21):

http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html#AppA

[udk06]   UNO development Kit project homepage, URL (2006-06-11):

http://udk.openoffice.org/

[Ull05]   Ullenboom, Christion. Java ist auch eine Insel. Galileo Press Gmbh, Bonn
             2005

[Wik06-1]   MD2. Wikimedia Foundation, Inc. URL (2006-06-21.):

             http://en.wikipedia.org/wiki/MD2

[Wik06-2]   MD5. Wikimedia Foundation, Inc. URL (2006-06-21.):

             http://de.wikipedia.org/wiki/MD5

[Wiki06-3]   XPath. Wikimedia Foundation, Inc. URL (2006-06-21):

             http://de.wikipedia.org/wiki/XPATH

[Wiki06]   Rexx. Wikimedia Foundation, Inc. URL (2006-06-22):

             http://en.wikipedia.org/wiki/Rexx

[wiki3D06]   Wikipedia – Java 3D, URL (2006-06-20)

             http://en.wikipedia.org/wiki/Java_3D