# Scripting Mozilla with JavaXPCOM and BSF4ooRexx

Seminar Paper

Gerald Rauter
matriculation number: 0852429

Vienna University of Economics and Business

31th July 2011

## Abstract English

This seminar paper tries to show how to script Mozilla components via XPCOM, JavaXPCOM, BSF4ooRexx and ooRexx. The used technologies are explained through summarizing text and Nutshell examples. The goal of this paper is to show the features and possibilities of this approach.



## Abstract Deutsch

Diese Seminararbeit versucht zu zeigen, wie man Mozilla-Komponenten mithilfe von XPCOM, JavaXPCOM, BSF4ooRexx und ooRexx skriptet. Die benutzten Technologien werden mithilfe von zusammenfassenden Texten und Nutshell-Beispielen erklärt. Das Ziel dieser Arbeit ist es die Funktionen und Möglichkeiten dieser Vorgehensweise aufzuzeigen.

# Table of content

# List of figures



# List of examples

# 1) Introduction

## *1.1) About this paper*

This seminar paper has been written for the fifth course of the specialization 'Management Information Systems' at the Vienna University of Economics and Business. Course lecturer is Prof. Rony G. Flatscher.

Chapter one gives a short introduction to the topic.

Chapter two is about the used software. This chapter will give a brief introduction of ooRexx, BSF4ooRexx, XPCOM and JavaXPCOM.

Chapter three contains a step to step installation guide of how to set up the right environment.

Chapter four contains some examples, which show what's possible with this topic and chapter five gives a short conclusion.

## *1.2) About the topic*

The original plan of this paper was to use XPCOM the same way like Microsoft COM would be used for Microsoft products, which means to remotely open the programs Mozilla Firefox, Thunderbird and SeaMonkey, work with them and close them again via a ooRexx Script.

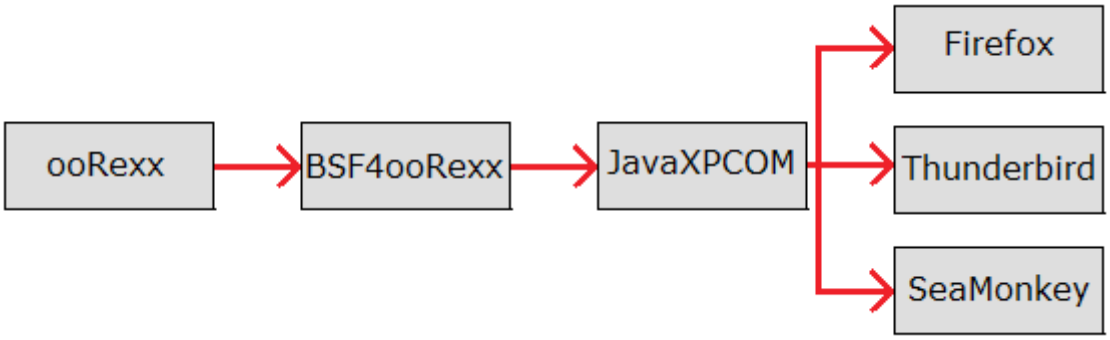


Figure 1.1: Original Plan of Action

Unfortunately XPCOM works differently than Microsoft COM. These differences will be addressed in a later chapter. Because of these circumstances the author wasn't able to script the whole programs, but 'only' managed to embed independent parts of the programs, so called components, into short examples.

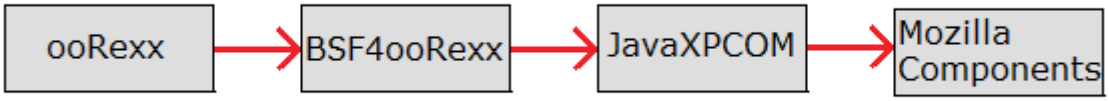


Figure 1.2: New Plan of Action

XPCOM makes it possible to work with these components of Mozilla Projects. These components are small, independent parts of programs, which have their own responsibilities. One component can be for example the bookmarkservice, which is responsible for creating, reading and deleting bookmarks.

*"JavaXPCOM allows for communication between Java and XPCOM, such that a*

*Java application can access XPCOM objects, and XPCOM can access any Java class that implements an XPCOM interface."* [Source: xpc1]

If it's possible to work with XPCOM via Java, it's of course also possible to use BSF4ooRexx to work with XPCOM via ooRexx, which will be done in this paper. More about ooRexx, BSF4ooRexx, JavaXPCOM and XPCOM will be explained in the next chapter.

# 2) Used Software

## *2.1) ooRexx*

Open Object Rexx (**RE**structured e**X**tended e**X**ecutor) is the open source, object orientated version of Rexx.
Rexx was developed by Mike Cowlishaw at IBM. It's an interpreted programming language (in contrast to compiled programming languages) which has an English-like syntax to make it easy to use, read and learn. Today there are many interpreters available for a wide range of computing platforms. [Source: wiki1]

IBM has release their Rexx and Object Rexx implementation's sources under the Common Public License. Nowadays it is further developed by the the Rexx Language Association (RexxLA) under the name Open Object Rexx. The Rexx Language Association (RexxLA) is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. [Source: rexla1]

The biggest advantages of Rexx is it's simple English-like syntax, it's dynamic data typing and it's small and thus easy to learn instruction set.

***An easy Rexx example:***

```
say "Hello World!"
```

An easy example. This program prints "Hello World!" on the screen.

## *2.2) BSF4ooRexx*

The **B**ean **S**cripting **F**ramework (BSF) allows other programming languages to interact with the Java runtime environment (JRE) and through this allows the use of Java classes and Java objects by them. [Source: flatscher1]

BSF4ooRexx is the Bean Scripting Framework for ooRexx. It makes it possible to use Java classes and objects with ooRexx programs and allows Java programs to use scripts which are written with ooRexx. With BSF4ooRexx nearly everything that can be done with Java can also be done with ooRexx, which has an easier syntax than Java.

BSF4ooRexx is developed by Mag. Dr. Rony G. Flatscher at the Vienna University of Economics and Business. That's why the most actual version is called the "Wiener Version". Older available versions are the "Essner Version" and the "Augsburger Version". These Versions are named after the cities of Universities Prof. Mag. Dr. Flatscher was working at while developing them. [Source: hoisl1]

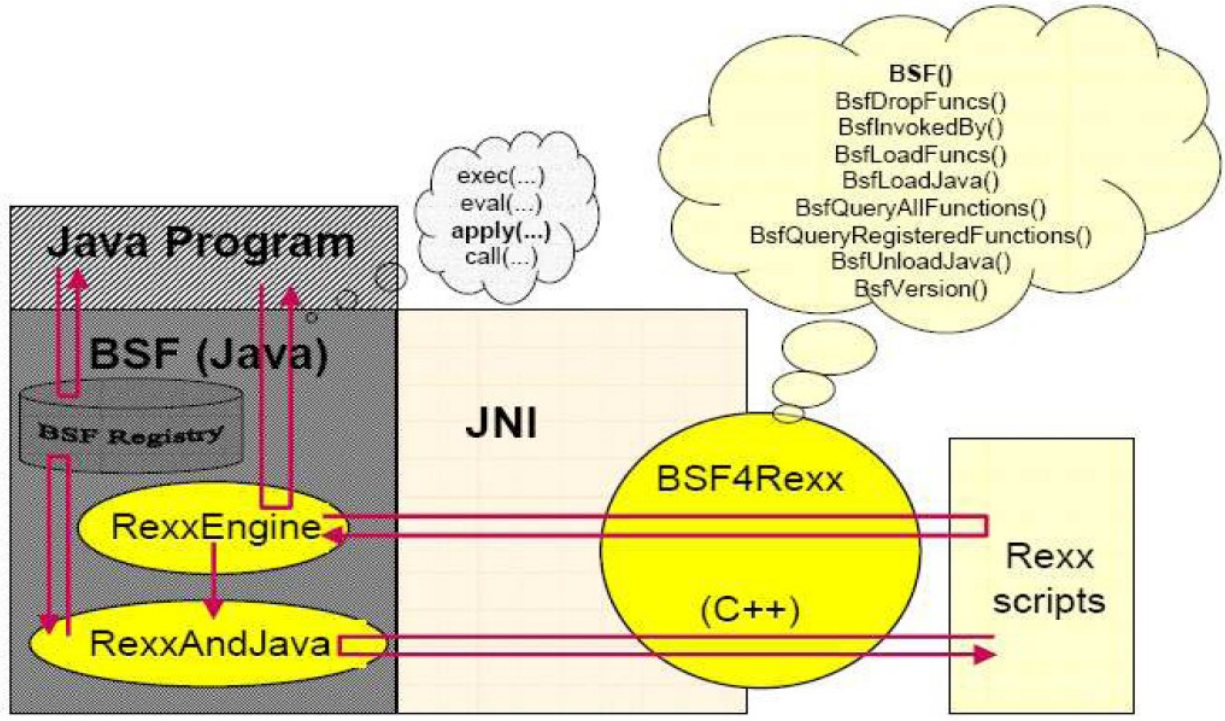


Figure 2.1: Architecture of the "Wiener Version" of BSF4ooRexx

## *2.3) XPCOM*

"The Cross Platform Component Object Module (XPCOM) is a framework which allows developers to break up monolithic software projects into smaller modularized pieces. These pieces, known as *components*, are then assembled back together at runtime." [Source: Dev5]
Because of its component based approach XPCOM makes software developing and software maintenance easier. Existing components can be put together with new components to create new programs. A developer doesn't have to program all features of a program by himself if existing components already provide needed features.
The main advantage of components are, that they can be reused by several programs and they can be easily replaced by other components, for example for upgrades.
Mozilla Projects like Firefox, Thunderbird, SeaMonkey and others are made out of several components. Some components are used by all of them, like window management, and some components are only used by one project.
Different components are linked together through interfaces. All components support the base interface nsISupports, which carries out administrative task like component lifetime management and interface querying. A list of all public XPCOM interfaces can be found at [Dev4]

XPCOM itself is written in C++, but there are APIs available to use it with many other programming languages. It is possible to use it with JavaScript, Java, Python, Perl and Ruby, if fitting language bindings are installed.

XPCOM is similar to Microsoft COM, but it's designed to be used mainly at the application level. XPCOM is Open Source, in contrary to MSCOM.

## *2.4) JavaXPCOM*

"JavaXPCOM provides a bridge for Java applications to embed Gecko and use XPCOM components."[Source: Dev3]

JavaXPCOM allows Java to communicate with XPCOM. With its help Java programs can talk to XPCOM respectively it's possible to embed XPCOM components into Java programs or, like in the case of this paper, into ooRexx programs with the help of BSF4ooRexx.

JavaXPCOM was part of the XULrunner installation until version 1.9.2, but it has been removed in version 2. [Source: Dev4]. Also, Mark Finkle, who works for the Mozilla Corporation, wrote in his blog on July 28th 2010 that "JavaXPCOM has been disabled and will likely to removed from the Mozilla source tree in the future." [Source: Finkle1]
This means that JavaXPCOM won't be further developed and won't be supported anymore.

# 3) Installation

## *3.1) Installation of ooRexx*

The newest versions of ooRexx can be found under
http://www.oorexx.org/download.html or
http://sourceforge.net/projects/oorexx/files/.
To run all the examples in this paper at least version 4.1 is needed. A graphical GUI guides through the installation of the program. The user just has to accept the license agreement, specify a directory where ooRexx should be installed to and what parts of ooRexx should be installed.

During the installation on a windows computer the windows service rxapi has to be installed and started. The installation of version 4.1 of ooRexx sometimes wants to stop a running rxapi, which doesn't exist and aborts the installation, because it can't be stopped. To avoid this problem an older version of ooRexx, like 3.2 can be installed first, which installs and starts the rxapi service. After that, the installation of 4.1 can be started. It de-installs the older ooRexx version, stops the now existing rxapi and installs itself without further problems.

## *3.2) Installation of BSF4ooRexx*

The newest graphical installer for Windows, Linux and Mac of BSF4ooRexx can be found under http://sourceforge.net/projects/bsf4oorexx/files/experimental/

The user has to accept the terms of the license agreement and specify an install-directory. After the installation is finished the user is able to use Java with ooRexx.

## *3.3) Installation of JavaXPCOM*

JavaXPCOM is part of the XULRunner version 1.9.2, which can be found under http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases/1.9.2.19/sdk/

Later releases of the XULRunner, like version 2 don't include JavaXPCOM anymore.

The user just has to unzip the .zip-file with an archive tool into a directory. It's not important, where its unpacked to, but Mozilla recommends the directory '`C:\Program Files\Mozilla XULRunner\1.9.2`' for Microsoft Windows or '`/opt/xulrunner/1.9.2`' for Linux.

To register XULRunner with the system, a command prompt has to be opened and '`xulrunner.exe --register-global`' (to register for all users) or '`xulrunner.exe --register-user`' (to register for one user only) has to be run.

With Linux the command is '`xulrunner –register-global`' respectively '`xulrunner –register-user`'

[Source: Dev6]

As a last step the path to the four .jar-archives 'javaxpcom.jar', 'MozillaInterfaces.jar', 'MozillaInterfaces-svc.jar' and 'MozillaGlue.jar' has to be added to the environment variable CLASSPATH. This four archives can be found in the installation directory of the XULRunner in the \bin directory respectively in the \lib directory.

With Windows the CLASSPATH can be set in the command window with the command: 'set CLASSPATH=%classpath%[Path of the .jar];'. The %classpath % is needed to add the new path to the existing ones. Without it the variable classpath would be overwritten. With Windows it's also possible to set the

environment variables through right click to Computer on the desktop\Configure\Advanced System Configuration\Environment Variables. With Linux the environment variable CLASSPATH can be set with the commands 'CLASSPATH=$CLASSPATH: /[Path of the driver.jar]'

'export PATH'.

After the CLASSPATH is set JavaXPCOM is ready for use.

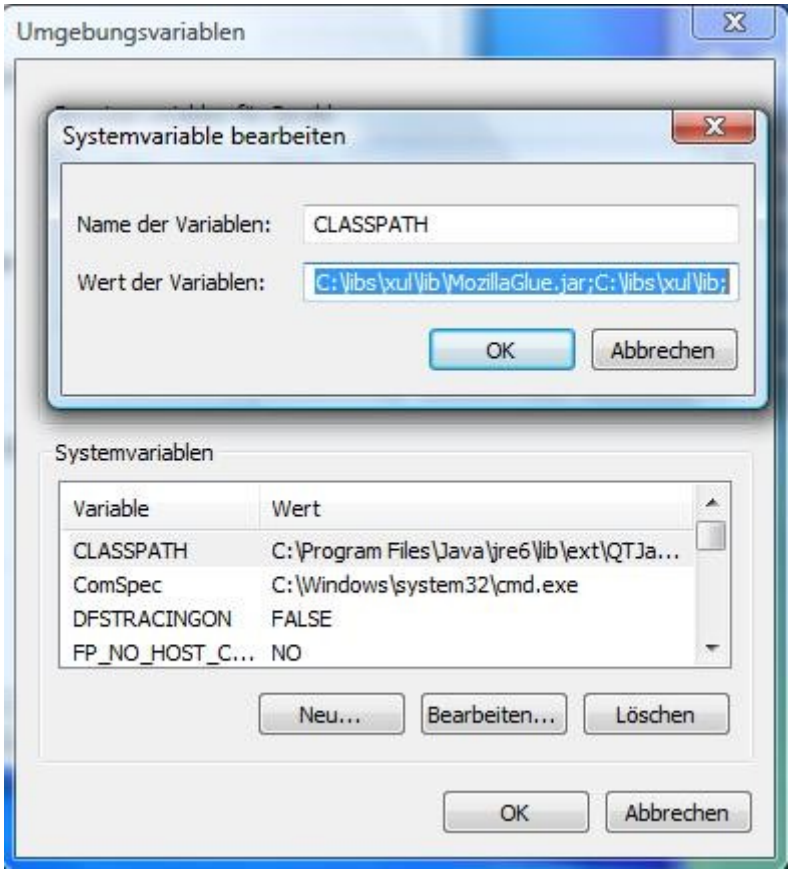


Figure 3.1: Set CLASSPATH

# 4) Examples

This chapter contains examples, which show what's possible to achieve with JavaXPCOM. Examples 4.1 and 4.2 are, with some small changes, taken out of the seminar paper of Martin Palkovic from 2010. Examples 4.4 and 4.5 are available in the same paper as java classes. They were, with some small changes, rewritten as .jrexx files for this paper. Also, the class LocationProvider.class, which is used in three examples was taken from Martin Palkovic's work [Source: Palk1]

All examples follow the same pattern. First, all needed classes are imported with the command:

```
.bsf~bsf.import('qualified.class.name','new name in this example')
```

In the next step, XPCOM is initialized. The installation path of the xulrunner is needed for initialization. There are two ways of getting the installation path. The first way, which is used in this paper, is to simply define it manually. In the following examples this is done by:

```
grePathName = "C:\libs\xul\xulrunner-sdk\bin"
grePath = .File~new(grePathName)
```

The other way, which is for example used by Martin Plakovic, is with the following command:

```
grePathName = .System~getProperty('GRE_PATH')
grePath = .File~new(grePathName)
```

This command gets the installation path of the xulrunner out of the system properties. However, if the registration of the installation is flawed, this command only retrieves a .nil object.

After the path is identified XPCOM can be initialized with the commands:

```
 mozilla~initialize(grePath)
 mozilla~initXPCOM(grePath, .nil)
```

Next, the service manager and every other needed services can be loaded and used.

## *4.1) Opening an URL in a window*

This example loads an URL and opens it in a new window

```
1  /* importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
5  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
6  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
7  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')
8
9  /* URL, that will be visited */
10 targetUrl = 'http://www.wu.ac.at'
11
12 /* setting the installation path of the xulrunner */
13 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
14 grePath = .File~new(grePathName)
15
16 /* initialize XPCOM */
17 mozilla = .Mozilla~getInstance
18 mozilla~initialize(grePath)
19 mozilla~initXPCOM(grePath, .nil)
20 say 'Mozilla XPCOM initialized!'
21
22 /* getting the serviceManager */
23 serviceManager = mozilla~getServiceManager
24
25 /* getting IIDs, which are needed to get the services */
26 appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
27 windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
28 windowWatcherID = .nsIWindowWatcher~NS_IWINDOWWATCHER_IID
29
30 /* setting properties of the window */
31 winProps = "width=1000, height=600, resizable, centerscreen, scrollbars='yes'"
32
33 /* getting the needed services */
34 appStartup = serviceManager~getServiceByContractID('@mozilla.org/toolkit/app-'-
       ||'startup;1', appStartupID)
35 windowCreator = appStartup~queryInterface(windowCreatorID)
36 WindowWatcher = -
       serviceManager~getServiceByContractID('@mozilla.org/embedcomp/window-'-
       ||'watcher;1', windowWatcherID)
37 windowWatcher~setWindowCreator(windowCreator)
38
39 /* opening the window */
40 window = windowWatcher~openWindow(.nil, targetUrl, 'BrowserWindow', winProps, -
       .nil)
41 windowWatcher~setActiveWindow(window)
42 appStartup~run
43
44 mozilla~shutdownXPCOM(.nil)
45 say 'Mozilla XPCOM embedding finished!'
46
47 ::requires BSF.cls
```

In lines 2 to 7 all needed classes for this example are loaded. The classes 'org.mozilla.xpcom.Mozilla' and java.io.File have to be loaded in every following example. They're needed to initialize XPCOM.
The classes loaded in lines 4 to 7 are the interfaces of the components, which are used in this example.
In line 10 the URL which should be loaded is saved in a variable. In this case it is www.wu.ac.at. In lines 13 and 14 the path to the xulrunner installation is saved in a variable. Like mentioned earlier it is needed for the initialization of XPCOM, which is done in lines 17 to 19. To be able to use XPCOM, this initialization has to be done in every example.
Line 23 loads a service manager, which is used to load and manage all other services.
To load other services their contractID and their interfaceID is needed. In lines 26 to 28 the interfaceIDs of the services are saved into variables for later use.
In line 31 the properties of the window which will be created are saved in a string. There are only a few possible properties used. A full list of possible properties can be found at [Dev1]. In lines 34 to 36 the interfaces of all other needed services are loaded. These services are AppStartup, WindowCreator and WindowWatcher. AppStartup is an service which helps to startup Applications, WindowCreator allowes to open new windows and WindowWatcher 'watches' over open windows and allows some operations on them. AppStartup and WindowWatcher are loaded with the servicemanager, while WindowCreator is loaded with the command ~queryInterface. When loaded with the servicemanager, an instance of the requested service is created, while when loaded with ~queryInterface only a pointer to the desired interface is returned. [Source: Palk1]
In line 37 the WindowCreator is assigned to the WindowWatcher.
Finally in lines 40 to 42 the new window is opened and displayed. The arguments for the ~openWindow message are: a parent window (.nil if there is no parent window), the URL which should be loaded, the name of the new window, the features of the window (which were saved in winprops earlier) and additional arguments. After the window is closed XPCOM is shut down and the

example is finished.

## *4.2) Reading and deleting Cookies*

This example will load an URL in a new window, load all cookies, show their name, host and value and delete them again.

```
1  /* importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
5  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
6  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
7  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')
8
9  .bsf~bsf.import('org.mozilla.interfaces.nsICookieManager','nsICookieManager')
10 .bsf~bsf.import('org.mozilla.interfaces.nsICookie','nsICookie')
11
12 /* URL, that will be visited */
13 targetUrl = 'http://derstandard.at'
14
15
16 /* Initiate XPCOM embedding */
17 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
18 grePath = .File~new(grePathName)
19 mozilla = .Mozilla~getInstance
20 mozilla~initialize(grePath)
21 mozilla~initXPCOM(grePath, .nil)
22 say 'Mozilla XPCOM initialized!'
23
24 /* getting the Service Manager */
25 serviceManager = mozilla~getServiceManager
26
27 /* getting IIDs, which are needed to get the services */
28 appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
29 windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
30 windowWatcherID = .nsIWindowWatcher~NS_IWINDOWWATCHER_IID
31 cookieManagerID = .nsICookieManager~NS_ICOOKIEMANAGER_IID
32 cookieID = .nsICookie~NS_ICOOKIE_IID
33
34 /* window propterites */
35 winProps = "width=1000, height=600, resizable, centerscreen, scrollbars='yes'"
36
37 /* getting the needed services */
38 appStartup = serviceManager~getServiceByContractID('@mozilla.org/toolkit/app-'-
       ||'startup;1', appStartupID)
39 windowCreator = appStartup~queryInterface(windowCreatorID)
40 WindowWatcher = -
       serviceManager~getServiceByContractID('@mozilla.org/embedcomp/window-'-
       ||'watcher;1', windowWatcherID)
41 windowWatcher~setWindowCreator(windowCreator)
42 CookieManager = -
       serviceManager~getServiceByContractID('@mozilla.org/cookiemanager;1', -
       cookieManagerID)
```

```
43 /* opening the window */
44 window = windowWatcher~openWindow(.nil,targetUrl,'BrowserWindow',winProps,.nil)
45 windowWatcher~setActiveWindow(window)
46 appStartup~run
47
48 /* count cookies and print their name, host and value on the screen */
49 enumerator = cookieManager~getEnumerator
50 do while enumerator~hasMoreElements
51 cookies=enumerator~getNext
52 cookie=cookies~queryInterface(cookieID)
53 say pp("Cookie Name:") cookie~getName
54 say "L" "09"x pp("Host:") cookie~getHost
55 say "L" "09"x pp("Value:")  cookie~getValue
56 end
57
58 /* remove all cookies */
59 cookieManager~removeAll
60 say 'All cookies deleted'
61
62 /* Terminate XPCOM embedding */
63 mozilla~shutdownXPCOM(.nil)
64 say 'Mozilla XPCOM embedding finished!'
65 ::requires BSF.cls
```

Lines 2 to 10 load the needed classes. Two more classes than in the previous example are loaded, which are the interface for the cookiemanager and the interface for cookies.

Again, a target URL is saved in a variable in line 13. This time it's the URL http://derstandard.at.

The initialization of XPCOM in lines 17 to 22 is the completely the same as in the previous example.

Also all the lines till line 48 are quite similar to the previous example. A service manager is loaded, the interfaceIDs of all used interfaces are saved in variables, the properties of the new window are saved into a string variable, the services are loaded and the new window is opened.

In line 49 the attribute 'Enumerator' of the cookie manager is read and saved in the variable enumerator. This attribute enumerates all cookies in the cookie list.

Lines 50 to 56 are a loop, which prints the names, hosts and values of all cookies on the screen.

After that all cookies are deleted in line 59 and XPCOM shuts down.

## *4.3) Creating and reading History entries*

In this example some URIs will be added to the browser history. After that, the browser history will be checked and the last page visited will be printed on the screen.

```
1  /* importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
5  .bsf~bsf.import('org.mozilla.interfaces.nsIIOService', 'IOService')
6  .bsf~bsf.import('org.mozilla.interfaces.nsIBrowserHistory', 'nsIBrowserHistory')
7
8  /* setting targetUrls */
9  targetUrl = 'http://www.wu.ac.at'
10 targetUrl2 = 'http://derstandard.at'
11 targetUrl3 = 'http://www.teamliquid.net'
12
13
14 /** Initiate XPCOM embedding with location provider */
15 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
16 grePath = .File~new(grePathName)
17 mozilla = .Mozilla~getInstance
18 mozilla~initialize(grePath)
19
20 locprovider=.bsf~new("LocationProvider", grePath)
21 mozilla~initXPCOM(grePath, locprovider)
22 say 'Mozilla XPCOM initialized!'
23
24 /* Get the Service Manager */
25 serviceManager = mozilla~getServiceManager
26
27 /* get needed IDs */
28 ioserviceID = .IOService~NS_IIOSERVICE_IID
29 browserhistoryID = .nsIBrowserHistory~NS_IBROWSERHISTORY_IID
30
31 /* getting the needed services */
32 ioservice = serviceManager~getServiceByContractID('@mozilla.org/network/io-'-
       ||'service;1', ioserviceID)
33 browserhistory = -
       serviceManager~getServiceByContractID('@mozilla.org/browser/nav-history-'-
       ||'service;1', browserhistoryID)
34
35 /* creating URIs */
36 uri = ioservice~newURI(targetUrl, .nil, .nil)
37 uri2 = ioservice~newURI(targetUrl2, .nil, .nil)
38 uri3 = ioservice~newURI(targetUrl3, .nil, .nil)
39
40 /* adding 2 URIs to the browsing history */
41 browserhistory~addURI(uri,0,1,.nil)
42 browserhistory~addURI(uri2,0,1,.nil)
```

```
43 /* checking, which URIs are in the browser history */
44 say 'Is 'targetUrl 'in the browserhistory?'; if(browserhistory~isVisited(uri))-
   then say pp(yes); else; say pp(no)
45 say 'Is 'targetUrl2 'in the browserhistory?'; if(browserhistory~isVisited(uri2))-
   then say pp(yes); else; say pp(no)
46 say 'Is' targetUrl3 'in the browserhistory?'; if(browserhistory~isVisited(uri3))-
   then say pp(yes); else; say pp(no)
47 say '------'
48 say 'Last page visited was:' browserhistory~lastPageVisited
49
50 /* terminate XPCOM embedding */
51 mozilla~shutdownXPCOM(.nil)
52 say 'Mozilla XPCOM embedding finished!'
53
54 ::requires BSF.cls
```

Lines 2 to 6 again load the needed classes. Some classes, which were used earlier aren't necessary in this example and two new classes will be used, which are the interfaces for IOService and BrowserHistory. IOService is needed to create URIs out of the targetURLs, because some messages to the interface BrowserHistory only accept URIs.
Lines 9 to 11 save some URLs in variables for later use.
Lines 15 to 21 initialize XPCOM. This time it's a little bit different than in the previous examples, because a Location Provider is needed.

---

## Location Provider

The browser history and the bookmarks are saved in an SQLite database with the name 'places'. This database is saved in the user's profile directory. To access files and directories, both local and remote, XPCOM needs a location provider. A location provider provides services of XPCOM with relevant paths, if they are needed. A location provider in form of an java class can be found in the appendix of this paper. This java class has to be in the same directory as the examples the path to it has to be written in the CLASSPATH variable to work. A ooRexx implementation of the location provider, named locationProvider.cls, can also be found in the appendix. This version of the

location provider was written by Mag. Dr. Rony G. Flatscher. A version of example 4.5, which uses the ooRexx location provider can also be found there. In all other examples the Java version of the location provider will be used.

If the location provider is in the same directory or in the CLASSPATH variable a instance of it can be created with the command:
locprovider=.bsf~new("LocationProvider", grePath).

This locprovider has to be the second argument of the command mozilla~initXPCOM(grePath, locprovider) which was .nil in earlier examples. Now XPCOM is initialized with an working location provider.

---

In the lines till line 33 the services will be loaded like always.

Lines 36 to 38 create URIs out of the targetURLs like earlier mentioned.

Lines 41 and 42 write the first two URIs in the browser history. This could also be done by visiting this sites.

Lines 44 to 46 check, if these URIs were visited once. The output for the first two will be 'yes', because they were just added 2 lines above and the output for the third URI will be 'no' if it wasn't visited though a window earlier.

In line 48 the URI of the last visited page will be printed on the screen. In this case it will be http://derstandard.at, because this was the last added URI.

After that XPCOM will be shut down and the example is finished.

## *4.4) Inserting Bookmarks*

This example will add an URI to the bookmarks and check the bookmarks for URIs.

```
1  /* importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
5
6  .bsf~bsf.import('org.mozilla.interfaces.nsINavBookmarksService', -
       'BookmarksService')
7  .bsf~bsf.import('org.mozilla.interfaces.nsIIOService', 'IOService')
8
9  /* setting pages, that will be bookmarked */
10 targetUrl = 'http://www.wu.ac.at'
11 targetUrl2 = 'http://derstandard.at'
12
13 /* setting the installation path of the xulrunner */
14 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
15 grePath = .File~new(grePathName)
16 mozilla = .Mozilla~getInstance
17 mozilla~initialize(grePath)
18
19 /* loading location provider */
20 locprovider=.bsf~new("LocationProvider", grePath)
21
22 /* Initiate XPCOM embedding */
23 mozilla~initXPCOM(grePath, locprovider)
24 say 'Mozilla XPCOM initialized!'
25
26 /* Get the Service Manager */
27 serviceManager = mozilla~getServiceManager
28
29 /* getting IIDs, which are needed to get the services */
30 bookmarkserviceID = .BookmarksService~NS_INAVBOOKMARKSSERVICE_IID
31 ioserviceID = .IOService~NS_IIOSERVICE_IID
32
33 /* getting the needed services */
34 bookmarkservice = -
       serviceManager~getServiceByContractID('@mozilla.org/browser/nav-bookmarks-'-
       ||'service;1', bookmarkserviceID)
35 ioservice = serviceManager~getServiceByContractID('@mozilla.org/network/io-'-
       ||'service;1', ioserviceID)
36
37 /* creating new folder in bookmarks */
38 menuFolder = bookmarkservice~getBookmarksMenuFolder
39 say 'creating new bookmarkfolder'
40 newFolderID = bookmarkservice~createFolder(menuFolder,'Uni', -
       bookmarkservice~DEFAULT_INDEX)
41
42 /* creating URIs */
43 uri = ioservice~newURI(targetUrl, .nil, .nil)
44 uri2 = ioservice~newURI(targetUrl2, .nil, .nil)
```

```
45 /* bookmarking targetUrl */
46 say 'bookmarking 'targetUrl
47 bookmarkservice~insertBookmark(newFolderID, uri, bookmarkservice~DEFAULT_INDEX,-
       'WU Home')
48
49 /* checking, if targetUrl and targetUrl2 are bookmarked */
50 say 'Is 'targetUrl 'bookmarked?'; if(bookmarkservice~isBookmarked(uri)) then -
       say pp(yes); else; say pp(no);
51 say 'Is 'targetUrl2 'bookmarked?'; if(bookmarkservice~isBookmarked(uri2)) then -
       say pp(yes); else; say pp(no);
52
53 /* terminate XPCOM embedding */
54 mozilla~shutdownXPCOM(.nil)
55 say 'Mozilla XPCOM embedding finished!'
56
57 ::requires BSF.cls
```

Lines 2 to 7 load all needed classes. The only new class in this example is nsINavBookmarksService, which is the service to work with bookmarks.

After that, two targetURLs are saved and XPCOM is initialized. Again a location provider is needed to be able to work with the SQLite database places, where the bookmarks are stored.

Then the interfaces for the services are loaded.

In line 38 the menu folder of the bookmarks is loaded and in line 40 a new folder with the name 'Uni' is created in this menu folder.

In lines 43 and 44 URIs are made out of the targetURLs, because the bookmarks service can only work with URIs.

In line 47 the first URI is added to the new folder in the bookmarks with the name 'WU Home'.

Lines 50 and 51 check if the two URIs are bookmarked. The first line will print out '[YES]', because it was bookmarked some lines above. The second line will print out '[NO]' if it wasn't bookmarked earlier.

After this XPCOM shuts down and the example is finished.

## *4.5) Saving a Website in a local file*

This example will load an URL, open it in a new window and save it in a local .html-file.

```
1  * importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
5  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
6  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
7  .bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')
8
9  .bsf~bsf.import('org.mozilla.interfaces.nsIIOService', 'IOService')
10 .bsf~bsf.import('org.mozilla.interfaces.nsIWebBrowserPersist', -
       'nsIWebBrowserPersist')
11 .bsf~bsf.import('org.mozilla.interfaces.nsILocalFile', 'nsILocalFile')
12
13 /* getting path to the script */
14 dirpath= .File~new("")~getAbsolutePath
15 dirpath = dirpath"\savedsite.html"
16 say dirpath
17
18 /* setting targetUrl */
19 targetUrl = 'http://www.wu.ac.at'
20
21
22 /** Initiate XPCOM embedding with location provider */
23 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
24 grePath = .File~new(grePathName)
25 mozilla = .Mozilla~getInstance
26 mozilla~initialize(grePath)
27
28 locprovider=.bsf~new("LocationProvider", grePath)
29 mozilla~initXPCOM(grePath, locprovider)
30 say 'Mozilla XPCOM initialized!'
31
32 /* Get the Service Manager */
33 serviceManager = mozilla~getServiceManager
34
35 /* getting IIDs, which are needed to get the services */
36 appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
37 windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
38 windowWatcherID = .nsIWindowWatcher~NS_IWINDOWWATCHER_IID
39 ioserviceID = .IOService~NS_IIOSERVICE_IID
40 webbrowserpersistID = .nsIWebBrowserPersist~NS_IWEBBROWSERPERSIST_IID
41 localfileID = .nsILocalFile~NS_ILOCALFILE_IID
```

```
42 /* setting window properties */
43 winProps = "width=1000, height=600, resizable, centerscreen, scrollbars='yes'"
44
45 /* getting the needed services */
46 appStartup = serviceManager~getServiceByContractID('@mozilla.org/toolkit/app-'-
       ||'startup;1', appStartupID)
47 windowCreator = appStartup~queryInterface(windowCreatorID)
48 WindowWatcher = -
       serviceManager~getServiceByContractID('@mozilla.org/embedcomp/window-'-
       ||'watcher;1', windowWatcherID)
49 windowWatcher~setWindowCreator(windowCreator)
50 ioservice = serviceManager~getServiceByContractID('@mozilla.org/network/io-'-
       ||'service;1', ioserviceID)
51 Webbrowserpersist = -
   serviceManager~getServiceByContractID('@mozilla.org/embedding/browser/'-
       ||'nsWebBrowserPersist;1', webbrowserpersistID)
52 localfile = serviceManager~getServiceByContractID('@mozilla.org/file/local;1',-
       localfileID)
53
54 /* creating URI */
55 uri = ioservice~newURI(targetURL, .nil, .nil)
56
57 /* create new local file */
58 localfile~initWithPath(dirpath)
59
60 /* save target of URI into local file */
61 webbrowserpersist~saveURI(uri, .nil, .nil, .nil, '', localfile)
62
63 /* open the window */
64 window = windowWatcher~openWindow(.nil,targetUrl,'BrowserWindow',winProps,.nil)
65 windowWatcher~setActiveWindow(window)
66 appStartup~run
67
68 /* terminate XPCOM embedding */
69 mozilla~shutdownXPCOM(.nil)
70 say 'Mozilla XPCOM embedding finished!'
71
72 ::requires BSF.cls
```

Lines 1 to 11 load all needed classes. All classes from the first example are loaded (because a window will be opened) as well as IOService and two new classes, WebBrowserPersist and LocalFile. WebBrowserPersist makes it possible to save documents to local files or to remote files and LocalFile makes it possible to access local files.

Lines 14 and 15 get the current path to the example file and add '\saved.html' to it. This will be the path to the new saved file. This way, the new html-file will be saved in the same location as the example file.

Lines 23 to 30 initialize XPCOM with a location provider.

In the lines until line 53 all needed services are loaded. This lines are quite

similar to all previous examples and won't be farther explained.
Line 55 again creates an URI out of the targetURL.
Line 58 initializes the local file and line 61 saves the webpage into that file.
After that the webpage is opened in a new window and the example is finished.

## *4.6) Playing sounds*

This example plays some sounds, namely the standard beep sound, the e-mail notification sound and a wav-file which is loaded from the internet.

```
1  /* importing needed classes */
2  .bsf~bsf.import('java.io.File','File')
3  .bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
4  .bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
5
6  .bsf~bsf.import('org.mozilla.interfaces.nsISound', 'nsISound')
7  .bsf~bsf.import('org.mozilla.interfaces.nsIURI', 'nsIURI')
8  .bsf~bsf.import('org.mozilla.interfaces.nsIURL', 'nsIURL')
9  .bsf~bsf.import('org.mozilla.interfaces.nsIIOService', 'IOService')
10
11 /* setting URL of the target sound */
12 SoundUrl = -
       'http://www.tote-taste.de/Einsatzleitung/Service/Wavs/material/WAV197.WAV'
13 --mp3 and midi don't work
14
15 /* Initiate XPCOM embedding */
16 grePathName = "C:\libs\xul\xulrunner-sdk\bin"
17 grePath = .File~new(grePathName)
18 mozilla = .Mozilla~getInstance
19 mozilla~initialize(grePath)
20 mozilla~initXPCOM(grePath, .nil)
21 say 'Mozilla XPCOM initialized!'
22
23 /*getting the service manager */
24 serviceManager = mozilla~getServiceManager
25
26 /* getting IIDs, which are needed to get the services */
27 soundID = .nsISound~NS_ISOUND_IID
28 ioserviceID = .IOService~NS_IIOSERVICE_IID
29 urlID = .nsIURL~NS_IURL_IID
30
31 /* getting the needed services */
32 sound = serviceManager~getServiceByContractID('@mozilla.org/sound;1', soundID)
33 ioservice = serviceManager~getServiceByContractID('@mozilla.org/network/io-'-
       ||'service;1', ioserviceID)
34
35 /* creating URI and converting it into an URL */
36 musikuri = ioservice~newURI(soundURL, .nil, .nil)
37 musikurl = musikuri~queryInterface(urlID)
```

```
38 /* play a windows beep, a e-mail-notification and the target sound */
39 sound~beep
40 SAY "a beep"
41 CALL SysSleep 1
42 sound~playEventSound(0)
43 SAY "e-mail sound"
44 CALL SysSleep 1
45 sound~play(musikurl)
46 SAY "birdsounds"
47 CALL SysSleep 8
48
49 /* terminate XPCOM embedding */
50 mozilla~shutdownXPCOM(.nil)
51 say 'Mozilla XPCOM embedding finished!'
52
53 ::requires BSF.cls
```

Lines 2 to 9 load all needed classes. Three classes, which weren't used until now are loaded, which are the interfaces for sound, URIs and URLs.

Line 12 saves the URL to the later played wav-file in the variable soundurl.

Lines 16 to 20 initialize XPCOM. This time no location provider is needed, that's why the second argument of the command in line 20 is .nil.

Lines 24 to 34 load all services like always.

Lines 36 and 37 convert the soundurl in an URI and back in an readable URL. This has to be done, because the value saved in the variable soundurl is only a string, but it has to be of the type nsIURL.

Line 39 plays the standard system beep.

Line 42 plays the e-mail-notification sound. The command ~playEventSound() accepts the name of the event as an argument, but also constant values. Zero (0) for example stands for the e-mail-notification sound. A full list of Events and constants can be found at [Dev2]..

Line 45 plays the sound of the musikurl. The argument has to be of the type nsIURL.

After the last sound XPCOM shuts down and the example is finished.

# 5) Conclusion

The goal of this paper was to show the possibilities of scripting Mozilla components via JavaXPCOM, BSF4ooRexx and ooRexx as well as explain how it's done with small nutshell examples.

OoRexx and BSF4ooRexx are fitting tools to work with JavaXPCOM to script Mozilla components. The easy syntax of ooRexx and it's dynamic data typing are great add-ons to the nearly infinite possibilities of the Java programming language.

Unfortunately Java and JavaXPCOM arn't the best tools to work with XPCOM. The lack of Java examples for this topic on the internet as well as the dropped support and the fact that JavaXPCOM isn't developed further by the Mozilla Corporation make it hard to work with this approach.

In the opinion of the author it's much easier and more rewarding to use C or C++ to work with XPCOM. XPCOM itself and most examples and add-ons on the internet are written in C++, which makes researching for this topic much easier.

However, if someone only knows ooRexx and Java the approach described in this paper is a great solution to work with XPCOM, but it's harder than working with C++.

# 6) Bibliography

[Palk1]: Martin Palkovic: JavaXPCOM: Mozilla Firefox Scripting. http://wi.wu-wien.ac.at/rgf/diplomarbeiten/Seminararbeiten/2010/201007_Palkovic/201007_Palkovic_JavaXPCOM_Mozilla_Firefox_Scripting.pdf last retrieved on 2011-07-28

[Dev1]: window.open, MDN Docs. https://developer.mozilla.org/en/DOM/window.open last retrieved on 2011-07-28

[Dev2]: Interface Reference for nsISound. https://developer.mozilla.org/en/XPCOM_Interface_Reference/nsISound last retrieved on 2011-07-20

[Dev3]: Building JavaXPCOM. https://developer.mozilla.org/en/Building_JavaXPCOM last retrieved on 2011-07-31

[Dev4]: XPCOM Interface Reference. https://developer.mozilla.org/en/XPCOM_Interface_Reference last retrieved on 2011-07-31

[Dev5]: An Overview of XPCOM. https://developer.mozilla.org/En/Creating_XPCOM_Components/An_Overview_of_XPCOM last retrieved on 2011-07-31

[Dev6]: XULRunner 1.9.2 Release Notes.

https://developer.mozilla.org/en/XULRunner_1.9.2_Release_Notes last retrieved on 2011-07-30

[xpc1]: JavaXPCOM. https://developer.mozilla.org/en/javaxpcom last retrieved on 2011-07-31

[wiki1]: Wikipedia: REXX http://en.wikipedia.org/wiki/REXX last retrieved on 2011-07-28

[rexla1]: The Rexx Language Association http://www.rexxla.org/rexxlang/ last retrieved on 2011-07-28

[flatscher1]: Flatscher, Rony G.: BSF4ooRexx http://wi.wu-wien.ac.at:8002/rgf/wu/lehre/autojava/material/foils/AutoJava-BSF4ooRexx-01.pdf last retrieved on 2011-07-30

[finkle1]:Mark Finkle: Mark Finkle's Weblog. http://starkravingfinkle.org/blog/2010/07/the-future-of-javaxpcom/ last retrieved on 2011-07-31

[hoisl1] Hoisl, Bernhard: Automating Subversion. Wien, 2005. http://wi.wu.ac.at/rgf/diplomarbeiten/BakkStuff/2005/200507_Subversion_Hoisl/200507_AutomatingSubversion.pdf last retrieved on 2011-07-30

# 7) Appendix

### *Source Code of the Java-Location Provider:*

This Location Provider was taken out of the seminar paper from Martin Palkovic [Source: Palk1]

LocationProvider.class has to be in the same directory as the examples or its path has to be added to the CLASSPATH variable.

```
import java.io.*;
import org.mozilla.xpcom.*;


public class LocationProvider implements IAppFileLocProvider {
private final File libXULPath;
int counter = 0;

public LocationProvider(File grePath) {
        this.libXULPath = grePath;
}

public File getFile(String aProp, boolean[] aPersistent) {
        File file = null;

        if (aProp.equals("GreD") || aProp.equals("GreComsD")) {
                file = libXULPath;
                if (aProp.equals("GreComsD")) {
                        file = new File(file, "components");
                }
        }
        else if (aProp.equals("MozBinD") || aProp.equals("CurProcD") ||
aProp.equals("ComsD") || aProp.equals("ProfD"))
        {
                file = libXULPath;
                if (aProp.equals("ComsD")) {
                file = new File(file, "components");
                }
        }
        return file;
}

public File[] getFiles(String aProp) {
        File[] files = null;
        if (aProp.equals("APluginsDL")) {
                files = new File[1];
                files[0] = new File(libXULPath, "plugins");
        }
        return files;
        }

}
```

### *Source code of the ooRexx location provider*

This version of the location provider was witten by Professor Rony G. Flatscher and sent to the author of this paper on the 2nd August 2011. The intended purpose of this location provider is to make it possible to address and work with JavaXPCOM only with ooRexx.

```
/*
   author:     Rony G. Flatscher (c) 2011
   purpose:    implement a LocationProvider in Rexx instead of Java by
               implementing
               the interface methods "org.mozilla.xpcom.IAppFileLocProvider"
   needs:      BSF4ooRexx
   date:       2011-08-02
   license:    AL 2.0 or CPL 1.0

-- usage-example (snippet):

-- ... cut ....
grePathName="C:\libs\xul\xulrunner-sdk\bin"     -- give path to xulrunner's bin
                                                -- directory
grePath=.bsf~new("java.io.File", grePathName)   -- create a File object

mozilla=.Mozilla~getInstance~~initialize(grePath)  -- create Mozilla instance
                                                   -- and initialize it

   -- create instance of Rexx class, supply 'grePath' File object
lp=.LocationProvider~new(grePath)               -- create a Rexx object

   -- create a Java Rexx proxy: allow it to be used in Java wherever a
   --"org.mozilla.xpcom.IAppFileLocProvider" is needed as an argument;
   -- all Java method invocations will be forwarded to the Rexx object:
rp=BSFCreateRexxProxy(lp,,"org.mozilla.xpcom.IAppFileLocProvider")

mozilla~initXPCOM(grePath,rp)    -- "rp" is the Java Rexx proxy forwarding
                                 --messages to its proxy, the "lp" Rexx object

-- ... cut ...
*/
```

```
/* Rexx implementation of the interface "org.mozilla.xpcom.IAppFileLocProvider".
*/

::class LocationProvider public

::attribute libXULPath  -- allow Rexx direct access to this attribute (just a
                        -- convenience)

::method '<init>'       -- Java constructor (just for completeness here)
  expose  libXULPath
  use arg libXULPath
  say "in '<init>' (Java constructor), received and memorized:" -
            libXULPath~toString

::method init           -- Rexx constructor
  expose  libXULPath
  use arg libXULPath
  say "in 'init' (Rexx constructor), received and memorized:" -
            libXULPath~toString

::method getFile        -- interface method implementation
  expose libXULPath
  use arg aProp, aPersistent

  file=.nil
  if wordpos(aProp,"ComsD CurProcD GreComsD GreD MozBinD ProfD")>0 then
  do
     file=libXULPath    -- point to directory
     if wordpos(aProp,"ComsD GreComsD")>0 then
        file=.bsf~new("java.io.File", file, "components")
  end
  return file

::method getFiles       -- interface method implementation
  expose libXULPath
  use arg aProp

  files=.nil
  if aProp="APluginsDL" then
     files=bsf.javaArrayOf(.bsf~new("java.io.File", libXULPath, "plugins"))

  return files
```

### *Example 4.5) with ooRexx location provider:*

```
/* importing needed classes */
.bsf~bsf.import('java.io.File','File')
.bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
.bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
.bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')

.bsf~bsf.import('org.mozilla.interfaces.nsIIOService', 'IOService')
.bsf~bsf.import('org.mozilla.interfaces.nsIWebBrowserPersist', -
      'nsIWebBrowserPersist')
.bsf~bsf.import('org.mozilla.interfaces.nsILocalFile', 'nsILocalFile')

/* getting path to the script */
dirpath= .File~new("")~getAbsolutePath
dirpath = dirpath"\savedsite.html"
say dirpath

/* setting targetUrl */
targetUrl = 'http://www.wu.ac.at'


/** Initiate XPCOM embedding with location provider */

grePathName="C:\libs\xul\xulrunner-sdk\bin"     -- give path to xulrunner's bin
                                                -- directory
grePath=.bsf~new("java.io.File", grePathName)   -- create a File object

mozilla=.Mozilla~getInstance~~initialize(grePath)  -- create Mozilla instance
                                                   --and initialize it

   -- create instance of Rexx class, supply 'grePath' File object
lp=.LocationProvider~new(grePath)               -- create a Rexx object

   -- create a Java Rexx proxy: allow it to be used in Java wherever a
   -- "org.mozilla.xpcom.IAppFileLocProvider" is needed as an argument;
   --all Java method invocations will be forwarded to the Rexx object:
rp=BSFCreateRexxProxy(lp,,"org.mozilla.xpcom.IAppFileLocProvider")

mozilla~initXPCOM(grePath,rp)    -- "rp" is the Java Rexx proxy forwarding
                              --messages to its proxy, the "lp" Rexx object
say 'Mozilla XPCOM initialized!'
```

```
/* Get the Service Manager */
serviceManager = mozilla~getServiceManager

/* getting IIDs, which are needed to get the services */
appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
windowWatcherID = .nsIWindowWatcher~NS_IWINDOWWATCHER_IID
ioserviceID = .IOService~NS_IIOSERVICE_IID
webbrowserpersistID = .nsIWebBrowserPersist~NS_IWEBBROWSERPERSIST_IID
localfileID = .nsILocalFile~NS_ILOCALFILE_IID

/* setting window properties */
winProps = "width=1000, height=600, resizable, centerscreen, scrollbars='yes'"

/* getting the needed services */
appStartup = serviceManager~getServiceByContractID('@mozilla.org/toolkit/app-'-
      ||'startup;1', appStartupID)
windowCreator = appStartup~queryInterface(windowCreatorID)
windowWatcher =   -
      serviceManager~getServiceByContractID('@mozilla.org/embedcomp/window-'-
      ||'watcher;1', windowWatcherID)
windowWatcher~setWindowCreator(windowCreator)
ioservice = serviceManager~getServiceByContractID('@mozilla.org/network/io-
service;1', ioserviceID)
webbrowserpersist = -
      serviceManager~getServiceByContractID('@mozilla.org/embedding/browser/'-
      ||'nsWebBrowserPersist;1', webbrowserpersistID)
localfile = serviceManager~getServiceByContractID('@mozilla.org/file/local;1',-
      localfileID)

/* creating URI */
uri = ioservice~newURI(targetURL, .nil, .nil)

/* create new local file */
localfile~initWithPath(dirpath)

/* save target of URI into local file */
webbrowserpersist~saveURI(uri, .nil, .nil, .nil, '', localfile)

/* open the window */
window = windowWatcher~openWindow(.nil,targetUrl,'BrowserWindow',winProps,.nil)
windowWatcher~setActiveWindow(window)
appStartup~run

/* terminate XPCOM embedding */
mozilla~shutdownXPCOM(.nil)
say 'Mozilla XPCOM embedding finished!'

::requires locationProvider.cls
::requires BSF.cls
```