JavaXPCOM:
Mozilla Firefox Scripting

Seminar Paper

Martin Palkovic

0351749

0675 IS-Projektseminar SS 2010

Ao. Univ.-Prof.

Mag. Dr. Rony G. Flatscher

Institute for Management Information Systems

Vienna University of Economics and Business Administration

Table of Contents

1 Mozilla Firefox

Mozilla Firefox is a free open source web browser. It is the second most widely used browser after commercial Microsoft Internet Explorer with market share of almost 24% in June 2010. [BMSh10]

The name Mozilla refers to both an Application Suite and a Cross-Platform Application Framework. Both are based on a code base released by Netscape under a free software open source license in 1998. Since then, it has been maintained by the Mozilla Foundation. The application suite contains, amongst others, the Firefox browser and the Thunderbird email client.

The Mozilla Firefox project was created by Dave Hyatt and Blake Ross as an experimental branch of the Mozilla project. First version of Firefox 1.0 was released on November 9, 2004 and the latest version is 3.6.6.

Firefox retains the cross-platform nature of the original Mozilla browser, using the XUL user interface markup language. It is possible to extend the browser's functions through the use of extensions and themes. This feature and a wide selection of third party add-ons have attracted many Firefox users.

The latest Firefox version provides functions as tabbed browsing, live bookmarking, download manager, spell checking, private browsing, incremental find, integrated search system and location-aware browsing, the last of which is based exclusively on Google services.

2 XPCOM

XPCOM, which stands for Cross Platform Component Object Model, is a framework for writing cross-platform, modular applications. [OvXP09] The basic idea of modularization is splitting monolithic software into several smaller pieces, known as components. These components are usually delivered in small, reusable binary libraries (a DLL on Windows, or a DSO on Unix). When there are two or more related components in a library, it is referred to as a module. The goal of XPCOM is to allow different pieces of software that offer different functionality to be developed and built independently of one another.

Breaking software into different components can help development and maintenance be less difficult. Component-based approach to programming has several advantages:

- Reuse: modular code can be reused in other applications and contexts.
- Updates: updating components without having to recompile the whole application.
- Performance: modules that are not needed necessary right away can be "lazy loaded" or not loaded at all. It can improve the performance of an application.
- Maintenance: modular software design can make specific parts of an application easier to find and maintain.

XPCOM enables developers to create components that can be reused in different applications or replaced to change the functionality of existing applications. It does not only support modular software development, it also provides functionality of a development platform, such as file and memory management, threads, basic data structures (strings, arrays, etc.), object message passing or component management. These core libraries and tools enable selective component loading and manipulation.

The most of the XPCOM components are not part of this core set and are available with other parts of the platform (Necko, Gecko) or with an application or even with an

extension. For example the networking module, known as “Necko”, is a library of bundled components for each of the network protocols (HTTP, FTP, etc.).

2.1 Gecko

“The most important use of XPCOM is within Gecko, an open source, standards compliant, embeddable web browser and toolkit for creating web browsers and other applications.” [OvXP09] It is the layout engine developed by the Mozilla Project, whose function is to read web content, such as HTML, CSS, XUL, and JavaScript, and render it on user's screen or print it. Gecko is used in many applications, such as Firefox, Thunderbird, Mozilla Suite, Camino, Flock, SeaMonkey, Netscape 9, etc.

Although, main client of XPCOM is Gecko, it can be used in many environments, which are unrelated to web browsing. However, this paper focuses on XPCOM components that provide web browsing functionality.

2.2 Language Bindings

XPCOM layer itself is written in C/C++. Therefore, its API can be accessed out-of-the-box using C or C++. In order to enable access to XPCOM API for other languages, additional language bindings are needed.

Such a language binding is a bridge between a certain language and XPCOM with purpose to: [LaBi09]

- Enable access to XPCOM objects from within that language (where access means reading/writing/creating XPCOM objects as well as calling methods on them).

- Expose modules written in the bound language as XPCOM objects, thereby enabling all other languages for which XPCOM bindings exist to access these modules.

Several language bindings exist for various languages:

JavaScript: XPConnect
Java: JavaXPCOM
Python: PyXPCOM
Perl: PlXPCOM
Ruby: RbXPCOM

Whereas XPConnect is part of Firefox and is actively used in XUL applications, JavaXPCOM is not part of Firefox and needs to be installed as a part of XULRunner or can be used in Eclipse applications via SWT.

2.3 Interfaces

The communication between different components takes place on the basis of formalized boundaries known as interfaces. Interfaces allow encapsulation of the implementation and inner workings of the software and allow clients to use the software without knowing how things are made.

In the component-based programming, it is important that components provide the same access to the same methods across different versions – that the interfaces, they provide, will be immutable and thus establish a contract with the clients that use the software. “In this respect, interface-based programming is often referred to as programming by contract”. [OvXP09]

Interfaces are written in a language called Cross Platform Interface Description Language (XPIDL). It allows any component to specify its interfaces in a way that all

other components can understand independently of their native language mechanisms. Hence, all Mozilla applications are composed of a large number of small reusable components, each exporting their interfaces in a way even JavaScript can use. Along with XUL, XPCOM thus provides a rich environment for extensions developers.

2.4 XPCOM vs. Microsoft COM

For those who might wonder whether XPCOM is like Microsoft COM, the short answer is yes, and no. Both platforms appear identical due to their common ideological heritage. They are interface based and all interfaces are derived from one base interface, which defines three methods: `QueryInterface` – used for interface dispensing, and `AddRef` and `Release` – used for lifetime management. Despite some similarities, MSCOM and XPCOM components are neither compatible with each other nor interchangeable. A specific glue code or wrapper is required for communication between components of the two frameworks. As an example, the embedding wrapper for Mozilla allows browser engine to appear as ActiveX of MSCOM, while it internally operates on XPCOM components. The differences between these technologies can be summarized in the following points.

First area of distinction is the component proxy support. A component proxy is a fake component, which is used to impersonate another component that cannot be accessed directly. MS COM supports an elaborate proxy mechanism that allows creation of components with different threading models or restrictions. A component can be either “in process” or “out of process”. [InXP01] Whereas the first runs inside an application, the latter runs as a separate application. A single threaded component has its own thread and other threads use a proxy for access. Apartment threaded components share a thread but still need a proxy. Free threaded components need no proxy mechanism for the same process. This built-in proxy mechanism in MSCOM provides an appropriate amount of thread safety for components. On the other hand, “XPCOM is tailored toward providing modular

component support at the application level, so proxy services only support multiple threads sharing an otherwise non-reentrant component." [InXP01] In other words, accessing a remote component requires creating a new proxy. XPCOM offers some components that help user to do this.

The second and more significant contrast between XPCOM and MS COM is the fact that XPCOM technology is open source. The XPCOM architecture is fully available for developers to inspect the libraries and trace and debug their own application code. Moreover, it is even possible to modify the code base and extend its functionality. In contrast, if MS COM developers have difficulties in understanding the way how components and libraries are created and loaded, they are at the mercy of whatever documentation is available. Admittedly, Microsoft has put great effort to promote their technology and provide good documentation. However, changes in Microsoft system-level libraries behavior can affect the behavior of components and applications that use them.

2.5 Interface Description

As it was already mentioned above, interfaces are described by an interface description language (IDL). XPCOM uses its own dialect of IDL, which is called Cross Platform Interface Description Language – XPIDL. It also provides an xpidl compiler, whose purpose is to create a type library file for each module. In addition, it has a feature of writing nearly all of the declaratory C++ code when starting a new project.

The following code shows an interface description file with a sample interface.

```
#include "nsISupports.idl"
[scriptable, uuid(f728830e-1dd1-11b2-9598-fb9f414f2465)]
interface nsIScreen : nsISupports {
void GetRect(out long left, out long top, out long width, out
    long height);
void GetAvailRect(out long left, out long top, out long width,
    out long height);
readonly attribute long pixelDepth;
```

```
readonly attribute long colorDepth;
};
```

Code example 1: Sample interface [XPCB01]

In the above example we can see that the name of the interface is `nsIScreen` and its base interface is `nsISupports`. This means that all methods and attributes defined for `nsISupports` are implicitly defined for `nsIScreen` as well. Additionally, the interface has two methods (`GetRect` and `GetAvailRect`) and each of them uses four outgoing parameters of type long. Similarly, it has also two attributes (`pixelDepth` and `colorDepth`) with `readonly` keyword, which informs that the attributes can be examined but not set. Attributes are distinguished from methods by the `attribute` keyword. There is an optional part of the interface description between the square brackets just above the interface name, which provides some useful metadata. The first keyword `scriptable` indicates that the interface is available to be used by JavaScript or other scripting languages. Next, the `uuid` key specifies the interface ID, which needs to be provided by a program that wishes to use the interface.

2.6 Interface Discovery

As it was described above, XPCOM uses interface-based approach to handle components. If a client code wants to use some functionality of a component, the interaction between them takes place strictly through the available interfaces. Most of the XPCOM components support several interfaces. For this reason, there is an interface dispensing mechanism, which is provided by `QueryInterface` method and offers functionality for managing interfaces, in particular: [SuXP01]

- Determining what interfaces are supported by a component
- Switching from one interface to another (and back again).

These two functions can be grouped together with a common name of interface discovery. Every XPCOM component is required to support a standard interface that handles interface discovery. The standard interface has to be the base interface, from which all other XPCOM interfaces are extended and thus providing additional methods and functionality and is called `nsISupports`.

```
interface nsISupports
{
void QueryInterface(in nsIIDRef uuid, out nsQIResult result);
nsrefcnt AddRef();
nsrefcnt Release();
};
```

Code example 2: Base interface nsISupport in a simplified IDL [SuXP01]

The first method of the base interface is `QueryInterface`, which is the one that actually is responsible for interface discovery. The following two methods `AddRef` and `Release` provide lifetime management of a component through reference counting.

The first parameter of `QueryInterface` method is a reference to a UUID (universally unique ID) number, which is 128 bits long and is written using hexadecimal digits. For example, the interface ID for `nsISupports` is: 00000000-0000-0000-c000-000000000046.

“This ID number specifies an interface that may or may not be supported by the component being queried.” [SuXP01] Either the component returns an error code, when an interface represented by a specific UUID is not supported, or it returns a successful result code and sets the second parameter to the address of the requested interface. It is important that software designers take special care when creating new interfaces and assigning them with unique interface IDs. An example,

how to query interfaces of a component using `QueryInterface` method, can be found in the code section of this paper.

2.7 Components Identification

Each component needs to be identified using one of two forms. One form of component specification is a 128 bit number called a component's class ID. The other form is a contract ID, which has a form of a text string. These forms are used by the component manager when requesting and creating components and either form is sufficient for their identification. The purpose of the contract ID is to guarantee a set of behavior and related interfaces to clients that want to use the component.

A contract ID has a recommended format, which looks as follows: [SuXP01]
"`@<internetdomain>/module[/submodule[...]];<version>[?<name>=<value>[&<name>=<value>[...]]]`".
And here is an example of a contract ID, in particular of the app-startup component class: `@mozilla.org/toolkit/app-startup;1`

For those who know MS COM, we can say that an XPCOM contract ID is functionally equivalent to a Program ID or ProgID.

2.8 Lifetime Management

It is necessary for components to keep count of how many outstanding interfaces have been issued. Lifetime management ensures that no component gets destroyed while a client code is attempting to use one of its interfaces. For this purpose, the base interface provides reference count. When an object dispenses another copy of its interface, the internal reference count increases and on the other hand, the reference count decreases when the interface is released. The object destroys itself, when its reference count drops to zero.

When the `QueryInterface` method queries a component, the `AddRef` method of the base interface is performed on that component and notifies the object that an interface pointer has been duplicated. When a client code stops using a component, it calls the `Release` method, which notifies the object that an interface pointer has been destroyed and the resources held by component for the client code can be released. Many XPCOM bugs can be traced to the problem of incomplete pairing of `AddRef`/`Release` methods in the client software. [SuXP01]

2.9 Component Manager and Service Manager

As the name of the component manager implies, it keeps records of currently installed components and what DLL or shared library has to be loaded when creating a component. A code can use services of the component manager using its interface `nsIComponentManager`, which provides methods to access factory objects and instantiates instances of classes.

XPCOM services are referred to as singleton objects. This means that multiple requests for a service will always receive an interface to the same component. [SuXP01] Interesting is the indirection, where the component manager itself works a service.

On the other hand, the service manager is responsible for loading and unloading services. “When a request is made for a service that is already loaded, the service manager is smart enough to return another pointer to the existing service instead of trying to construct another object.” [SuXP01]

2.10 Criticism

Despite the advantages of XPCOM and its component-based application development, this technology brings also some disadvantages with it. Probably the most serious issues with XPCOM based systems are the extensive amount of code and the well-known memory leaks. As Mitschek points out, “there is a lot of code involved in managing and monitoring all the components and objects, which could lead to a decrease in performance, especially in large applications.“ [Mits10] This was also the reason why Apple has decided to use KHTML to create their WebKit engine for their Safari browser instead of XPCOM’s Gecko layout engine. An example here is the Firefox browser, which has been criticized by a number of users for its excessive memory usage. [RMUF10]

The current goal of the Gecko developers is to reduce excessive use of XPCOM in Gecko. “The basic idea is to refactor interfaces to remove unnecessary ‘XPCOM style’ ugliness and other interface design errors.” [GeDC10] An Interface Style Guide for Gecko was created to help create efficient, easy-to-use interfaces. The idea of creating clean interface design is referred to as “deCOMtamination”.

3 JavaXPCOM

JavaXPCOM is a language binding that allows for communication between Java and XPCOM. It enables Java applications to call XPCOM objects, and XPCOM to use any Java class that implements an XPCOM interface. With JavaXPCOM, developers can also embed Gecko layout engine in their Java applications. It is very similar to XPConnect, which is the JavaScript-XPCOM bridge and uses XPIDL for interface description.

In order to make use of JavaXPCOM, it needs to be installed and it also requires an installed JDK. It does not come with Firefox application, but it is built by default as part of XULRunner, “which is development framework and runtime environment for JavaXPCOM components“[Mits10] and provides necessary interfaces to communicate with the XPCOM architecture.

3.1 XULRunner Installation

As of July 2010, the current version of XULRunner is 1.9.2 and corresponds with the current 3.6 version of Mozilla Firefox. Versions for Windows, Mac OS and Linux can be downloaded from the Mozilla website[1]. The downloaded SDK package should be unzipped in a user pre-defined directory. Then, the XULRunner has to be registered with the system. This can be done by executing one of the following commands in the command line: xulrunner --register-global for registration for all users or xulrunner --register-user for registration for current user only.

It is important to ensure that the path to the XULRunner installation directory (GRE path) is included in the path variable and also following archives are added to the classpath variable:

[1] http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases

- javaxpcom.jar
- MozillaInterfaces.jar
- MozillaInterfaces-src.jar
- MozillaGlue.jar

These paths are required to get a dynamic access to the JavaXPCOM functions and interface libraries. Moreover, the GRE path plays a crucial part in initiating the XPCOM embedding. [Mits10]

3.2 Java Example 1 – WindowCreator

In this section, we are going to create a simple JavaXPCOM application that loads a URL and shows the content in a new window. This window is not a Firefox window. It is a simple frame, which shows only the content of the downloaded web page. Getting more features for our “browser” would require more complex application that would implement additional components.

Like every other Java application that wants to use XPCOM, it is necessary to initiate it first. Following lines of code will show how to start XPCOM so we can use its functionality.

First of all, we need to import all necessary classes. Besides the `java.io` package, we will also need classes from `org.mozilla.xpcom` and `org.mozilla.interfaces` packages. Whereas the first one enables the set up of the XPCOM environment, the latter provides interfaces to components that we want to use in our application.

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsIAppStartup;
import org.mozilla.interfaces.nsIDOMWindow;
import org.mozilla.interfaces.nsIServiceManager;
import org.mozilla.interfaces.nsIWindowCreator;
```

```
import org.mozilla.interfaces.nsIWindowWatcher;
```

Code example 3: JavaXPCOM - Classes import

Next, we need to set the path to the XULRunner directory. JavaXPCOM contains code that is able to find its currently installed versions and set the path for the Gecko Runtime Environment. In the following step, we can instantiate and initialize the Mozilla singleton class using a valid GRE path. “XPCOM embedding is complete when either `mozilla.initXPCOM` or `mozilla.initEmbedding` is called.” [Mits10]

```
GREVersionRange[] range = new GREVersionRange[1];
range[0] = new GREVersionRange("1.8", true, "1.9+", true);
File grePath = null;
grePath = Mozilla.getGREPathWithProperties(range, null);

Mozilla mozilla = Mozilla.getInstance();
mozilla.initialize(grePath);
mozilla.initXPCOM(grePath, null);
```

Code example 4: JavaXPCOM – Mozilla initialization

Now, when XPCOM is initialized, we need to create instance of the Service Manager in order to use XPCOM services. In general, there are two ways of getting a service: either requesting it using the Service Manager and providing the right Contract ID and the Interface ID, or using `QueryInterface` method and the IID. While the first one creates instances of the requested services, the latter returns only a pointer to the desired interface. In our example we use both methods. We need following interfaces:

- `nsIAppStartup` – provides application startup and quitting services,
- `nsIWindowCreator` – callback interface used by Gecko to create new windows,
- `nsIWindowWatcher` – maintains the list of the top-level windows and allows some operations on them,
- `nsIDOMWindow` – represents a single window object that may also contain child windows.

```
nsIServiceManager serviceManager =
mozilla.getServiceManager();

nsIAppStartup appStartup =
    (nsIAppStartup)serviceManager.getServiceByContractID
    ("@mozilla.org/toolkit/app-startup;1",
    nsIAppStartup.NS_IAPPSTARTUP_IID);

nsIWindowCreator windowCreator =
    (nsIWindowCreator)appStartup.queryInterface
    (nsIWindowCreator.NS_IWINDOWCREATOR_IID);

nsIWindowWatcher windowWatcher =
    (nsIWindowWatcher)serviceManager.getServiceByContractID
    ("@mozilla.org/embedcomp/window-watcher;1",
    nsIWindowWatcher.NS_IWINDOWWATCHER_IID);
```

Code example 5: JavaXPCOM – Getting services

Now, when we have instantiated objects for all necessary services, we can start creating the window of our application. As the first step, we need to initialize the `windowWatcher` by implementing a `windowCreator` object. Next, we create an instance of `nsIDOMWIndow`, which represents the browser window. The arguments provided to `openWindow` method include: a parent window represented by another `nsIDOMWindow` object, a URL that is to be loaded, name of the window, window features and some extra arguments. We set the `activeWindow` property of the `windowWatcher` to our new window object `win` and start the application. The `run()` method of the `nsIAppStartup` interface hands over the application to xpcom/xul and blocks the Java program until all created windows are closed. “To prevent this you would have to implement multithreading by using the XPCOM interface `nsIEventQueue`.” [Mits10] At the end we need to terminate the use of XPCOM in our application by calling `shutdownXPCOM()` method, or by calling `termEmbedding()` if we have used `initEmbedding()` before. This will release resources used by XPCOM.

```
windowWatcher.setWindowCreator(windowCreator);
nsIDOMWindow win = windowWatcher.openWindow(null, targetURL,
    "mywindow", "chrome,resizable,centerscreen", null);
windowWatcher.setActiveWindow(win);
appStartup.run();

mozilla.shutdownXPCOM(null);
```

Code example 6: JavaXPCOM – Creating window

The above example shows us how to use XPCOM components via JavaXPCOM interfaces. We found out how to load a URL in a simple window. This window, however, does not provide any other functionality. As already mentioned above, additional functions can be implemented by calling respective services. In the next section we will discuss the bookmarks and history service provided by XPCOM.

3.3 Places

'Places' was introduced in Firefox 3 as API for the bookmark and history management with new features like annotations and favicons. It also offers more complex querying to make handling the places easier and more convenient. [MoPl10] Places stores all the bookmarks and history data in an SQLite database. For manipulation with the database it uses mozIStorageService interface. This database is stored in the user's profile directory.[2]

In order to be able to use some XPCOM services correctly, it is necessary to provide XPCOM during initialization with an implementation of a Location Provider that will return relevant path(s) and file(s) for the specific services. This is also the case for Places services. It allows XPCOM developers to choose the locations for data of various types like profiles, plug-ins, preferences, and so on. If XPCOM is not provided with any Location Provider, a null object will set the default behavior. However, not providing any implementation of such a class can lead to an

[2] On Windows XP the profile directory for Mozilla Firefox is: C:\Documents and Settings\Owner\Application Data\Mozilla\Firefox\Profiles\

unexpected error during the execution of the program. The Service Manager has a problem with getting specific services without any Location Provider, as the result it tries to return appears to be huge enough to throw an OutOfMemory Error. A sample implementation of the Location Provider can be found in the Appendix. The following examples will show how to get access to the Places services.

3.4 Java Example 2 – BookmarksManager

This example shows how to work with bookmarks in XPCOM. The initialization and termination of XPCOM is the same as in the previous example. However, there is one exception, which will be explained shortly. It creates a folder in the Bookmarks Menu and checks whether a bookmark exists with the specified URL. If it does not exist, program adds it into the created folder.

Here again, we have to import all necessary interface classes which we intend to use. The `nsINavBookmarksService` interface allows for accessing bookmarks database and manipulation with them. We will also need `nsIURI` – the uniform resource identifier interface and `nsIIOService`, which provides a set of network and URL parsing utility functions. Before we start initialize XPCOM, we create an instance of the above mentioned `LocationProvider` class, which will be used as a parameter for initialization. In our implementation of the `LocationProvider` the profile directory path is set to XULRunner installation directory. If the bookmarks or history management interfaces do not find the “places.sqlite” file in the directory, they will create one.

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsINavBookmarksService;
import org.mozilla.interfaces.nsIIOService;
import org.mozilla.interfaces.nsIURI;

LocationProvider locProvider = new LocationProvider(grePath);
mozilla.initXPCOM(grePath, locProvider);

nsINavBookmarksService bms = (nsINavBookmarksService)
```

```
    serviceManager.getServiceByContractID
    ("@mozilla.org/browser/nav-bookmarks-service;1",
    nsINavBookmarksService.NS_INAVBOOKMARKSSERVICE_IID);
nsIIOService ios = (nsIIOService)serviceManager
    .getServiceByContractID("mozilla.org/network/io-service;1",
    nsIIOService.NS_IIOSERVICE_IID);
```

Code example 7: BookmarksManager – Getting bookmarks service

Next, we need to get an ID of the parent folder, in which we will create a new folder using three parameters: name of a parent directory, name and index of the new directory. Setting the parameter to `DEFAULT_INDEX` will insert the folder on the last position within the parent folder. We can see that also the `createFolder()` method returns a variable of type long. This is because Places database, which stores bookmarks and history, uses URIs and integers for identification of the items. As the next step, we create a new URI object and check, whether there is a bookmark with this URI in the target folder. If it does not exist, the `insertBookmark()` method will add it to the created folder under the specified name.

```
long menuFolder = bms.getBookmarksMenuFolder();
long newFolderID = bms.createFolder(menuFolder,
    "search engine", bms.DEFAULT_INDEX);

nsIURI uri = ios.newURI("http://google.com/", null, null);
if (bms.isBookmarked(uri)) {
System.out.println(uri .getPrePath() + " has been bookmarked");
} else
bms.insertBookmark(newFolderID, uri, bms.DEFAULT_INDEX,
    "Google");}
```

Code example 8: BookmarksManager – Creating a folder and a bookmark

3.5 Example 3 – SaveToFile

This example shows how to save a URL to a local file. As usual, we need to import all necessary classes. After initializing XPCOM, we use the Service Manager to provide us with the service of the `nsIWebBrowserPersist` interface, which is used

to persisting URIs and DOM documents to a local or remote storage. Similarly we create instances of `nsIIOService` and `nsILocalFile` interfaces.

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsIIOService;
import org.mozilla.interfaces.nsIServiceManager;
import org.mozilla.interfaces.nsIWebBrowserPersist;
import org.mozilla.interfaces.nsILocalFile;
import org.mozilla.interfaces.nsIURI;

nsIWebBrowserPersist persist = (nsIWebBrowserPersist)
    serviceManager.getServiceByContractID
    ("@mozilla.org/embedding/browser/nsWebBrowserPersist;1",
    nsIWebBrowserPersist.NS_IWEBBROWSERPERSIST_IID );
nsIIOService ios = (nsIIOService)serviceManager.
    getServiceByContractID("mozilla.org/network/io-service;1",
    nsIIOService.NS_IIOSERVICE_IID);
nsILocalFile file = (nsILocalFile)serviceManager.
    getServiceByContractID("@mozilla.org/file/local;1",
    nsILocalFile.NS_ILOCALFILE_IID);
```

Code example 9: SaveToFile – Getting services

In the rest of the code we create a new URI and create new file with the `initWithPath()` method and path parameter. Finally, the `saveURI()` method saves the resource to the created file.

```
nsIURI uri = ios.newURI("http://google.com/", null, null);
file.initWithPath("C:/newFile.html");
persist.saveURI(uri, null, null, null, "", file);
```

Code example 10: SaveToFile – Save URI to a file

4 Scripting XPCOM with ooRexx

We can now create simple java applications that can access XPCOM components and make use of their functionality. However, one may want to control java applications and use this functionality within a scripting language. In this chapter will show how to use Java and XPCOM technology with ooRexx, a powerful and yet easy-to-use scripting language.

4.1 ooRexx

Open Object Rexx (ooRexx) “is an enhancement of classic Rexx; a powerful, full-featured programming language which has a human-oriented syntax“.[OORe09] It is an Open Source project managed by Rexx Language Association (RexxLA) and is distributed under Common Public License v1.0 allowing a free of charge implementation.

The ooRexx offers several advantages. It is a procedural language that implements all concepts of object-oriented programming, which can help to solve many problems very effectively. [Mits10] It provides a standardized API that allows high-level programming languages to use ooRexx methods, and allows using features that are more natural in those languages. It is also a language, which is very easy to use and learn. There are relatively few rules for code formatting and many of its instructions are meaningful words (e.g. say, pull, do…end, exit…). Moreover, ooRexx is referred to as a ‘typeless’ language, in which no type declaration is needed since all variables are treated as objects. In addition, it has a rich set of built-in functions and methods and provides a powerful string handling functionality.

Although ooRexx has facilities to write robust large applications, it also enables to write programs with minimum of overhead. It is mainly used for automation of applications and operating systems like Windows or Linux.

4.2 Installation

As of July 2010, the latest version of ooRexx is 4.0.1. It is available for Windows, Linux, AIX and MacOS. It has a simple installer, which is available to download along with documentation and source code from the download section of the Open Object Rexx website. [3]

We have learned that with ooRexx we can build simple and yet powerful applications. As the next step, it is necessary to introduce a bridge that will allow for communication between ooRexx and Java, called BSF4ooRexx.

4.3 BSF4ooRexx

“The Bean Scripting Framework (BSF) is a set of Java classes, which provide scripting language support within Java applications. It also provides access to Java objects and methods from supported scripting languages.” [JBSF10]

BSF supports scripting languages like JavaScript, Perl, Python and others. However, ooRexx needs a specific BSF engine called BSF4ooRexx. It is an extension to Rexx and ooRexx that consists of set of Java classes and an external function package. Similarly, BSF4ooRexx enables Java programs to invoke ooRexx scripts and vice versa. “This way all of Java can be viewed as a huge external Rexx function library from the perspective of Rexx, available on any platform Rexx is available“. [Flat03] As it is pointed out, ooRexx is an object-oriented interface to Java and so it can reduce its complexity. [Flat04]

[3] http://www.oorexx.org/download.html

4.4 Installation

The installation package of the current version of BSF4ooRexx is available to download from the website of Vienna University of Economics and Business Administration[4]. Before installing the new version of BSF4ooRexx, it is necessary to make sure that the old version is uninstalled properly. If an OpenOffice support was also installed with previous version, this has to be removed first. The uninstallation is carried out by running the following scripts from the Windows command line in this order:

1. uninstallOOo.cmd
2. uninstallBSF.cmd

In Linux, there are following scripts to be executed:

1. uninstallOOo.sh
2. uninstallBSF.sh

Next, we unzip the installation package and change into the created subdirectory “install”. Execution of “rexx setupBSF.rex” will create platform customized installation scripts. On Windows it is “installBSF.cmd” and on Linux “installBSF.sh”. Running the corresponding script will add the “bsf-rexx-engine.jar” and “bsf-v400-20090910.jar” (name depending on the current version) archives to the system’s classpath variable. Installation can be tested by invoking the script “infoBSF.rexx”. If installation was successful, the script returns information about installed ooRexx, BSF4ooRexx and Java.

When we have already installed ooRexx and its BSF engine, we can move on to practical examples. In the following chapter we will show how to work with ooRexx and address Java and JavaXPCOM.

[4] http://wi.wu-wien.ac.at/rgf/rexx/bsf4oorexx/current/

4.5 ooRexx Example 1 – WindowCreator

In this section we are going to reproduce the first Java example from chapter 3.2, which loads a URL and opens the content in a new browser window. The steps we carry out are the same like in the Java implementation.

Also in the ooRexx implementation we need to import the required Java classes. We need exactly the same classes and in Java. However, here we need to specify explicitly, which class to import since importing a whole package is not possible.

```
.bsf~bsf.import('java.io.File','File')
.bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
.bsf~bsf.import('org.mozilla.xpcom.GREVersionRange','GREVersionRange')
.bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
.bsf~bsf.import('org.mozilla.interfaces.nsIDOMWindow','nsIDOMWindow')
.bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')
```

Code example 11: ooRexx – Classes import

Here again, we need to set the path to the XULRunner installation directory. Mitschek points out a potential issue with XPCOM initialization. [Mits10] With a wrong GRE registration the `getGREPathWithProperties()` method may return a nulltype instead of a correct path to javaXPCOM.jar archive. He suggests a solution to look up for the system property that we had set up during the installation of XULRunner. Another solution is to simply set the path manually. Additionally, creating and setting the `grePath` and `targetURL` shows the simplicity of manipulation with typeless variables.

```
targetUrl = .bsf~bsf.lookupBean('targetUrl')
if targetUrl = .nil then targetUrl = 'http://www.wsj.com'
grePathName = "C:\Program Files\xulrunner"
path = .System~getProperty('GRE_PATH')
if path = .nil then grePath = .File~new(grePathName)
else grePath = .File~new(path)

mozilla = .Mozilla~getInstance
```

```
mozilla~initialize(grePath)
mozilla~initXPCOM(grePath, .nil)
say 'Mozilla XPCOM initialized!'
```

Code example 12: ooRexx – Setting the GRE path and initializing Mozilla

The following code gets the Service Manager and retrieves necessary property values and interface IDs. There are two ways to do this. In the first one we use `getStaticValue()` method to get the value of the `NS_IWINDOWWATCHER_IID` property of the `nsIWIndowWatcher` interface. The other one is simpler, in which we send a message with the name of the property to the interface object.

```
serviceManager = mozilla~getServiceManager
appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
windowWatcherID = .bsf~bsf.getStaticValue(.nsIWindowWatcher,
    'NS_IWINDOWWATCHER_IID')
winProps = "width=1000, height=600, resizable, centerscreen,
    scrollbars='yes', status='yes'"
```

Code example 13: ooRexx – Retrieving properties

The rest of the code is almost identical with the one of Java example. We use Service Manager as well as `queryInterface()` method to get the component services using the CIDs and the IIDs retrieved by the code above. Then we initialize `windowWatcher` by providing it with `windowCreator` instance. Next, a new window object is created and set as active. Finally, the application starts and opens the window with the URL. Also in this implementation when the program is finished with XPCOM, the initialization needs to be terminated. The very last line of the code represents a directive for Rexx interpreter that the program uses Java wrapper in order to load and manipulate with the Java classes.

```
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)
window = windowWatcher~openWindow(.nil, targetUrl, 'myWindow',
winProps, .nil)
windowWatcher~setActiveWindow(window)
appStartup~run

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

::requires BSF.cls
```

Code example 14: ooRexx – Creating window

4.6 ooRexx Example 2 – CookieManager

This is another example of an ooRexx program that uses XPCOM functionality. It builds on the previous example. This program implements an additional service of cookies management and shows, how we can read the cookies, get their properties and delete them.

To be able to work with cookies, we need to implement following interfaces: `nsICookieManager` and `nsICookie`. So we need to import the respective Java interface classes. After XPCOM gets initialized, we can retrieve the interface IDs and get the service of the Cookie Manager.

```
.bsf~bsf.import('org.mozilla.interfaces.nsICookieManager',
    'cookieManager')
.bsf~bsf.import('org.mozilla.interfaces.nsICookie','cookie')

cookieManagerID = .bsf~bsf.getStaticValue(.nsICookieManager,
    'NS_ICOOKIEMANAGER_IID')
cookieManager = serviceManager~getServiceByContractID
    ('@mozilla.org/cookiemanager;1', cookieManagerID)
cookieID =
    .bsf~bsf.getStaticValue(.nsICookie,'NS_ICOOKIE_IID')
```

Code example 15: CookieManager – Getting the cookie management service

Now we can call enumerator through each cookie in the cookie list. The enumerated objects are of type `nsICookie`. We create a loop and iterate over all elements, for which we dynamically get the `nsICookie` interface in order to be able to read their properties. The second loop is intended to check, whether the `removeAll` method has deleted all cookies from the list. If the `hasMoreElements` variable is false, it means all cookies were deleted.

```
iter = cookieManager~getEnumerator
do while iter~hasMoreElements
    cookies=iter~getNext
    cookie=cookies~queryInterface(cookieID)
    say cookie~getName
    say " "cookie~getHost
end

cookieManager~removeAll
iter2 = cookieManager~enumerator
if iter2~hasMoreElements then
    say 'Cookies not deleted' else
    say 'All cookies deleted'
```

Code example 16: CookieManager – Enumerator

Conclusion

This paper was aimed at introducing the XPCOM technology, its language bindings and the possibilities to access its web browsing services from Java. It is the open source feature of this framework and the possibility to freely use, modify and further develop the code base that enabled many developers around the globe to create a large number of extensions and thus enhancing the functionality Firefox already provides. This is one of the reasons, why it is the second most common used web browser and its market share constantly increases.

We have learned that XPCOM provides numerous language bindings including Java. JavaXPCOM allows any Java application to access and create XPCOM components. With the help of few examples it was presented how to make use of some specific services. Moreover, we have learned that there is even simpler way to use this functionality with a scripting language called ooRexx. With the bean scripting framework it represents an easy-to-use tool to control Java and XPCOM via JavaXPCOM.

Literature

[BMSh10] Browser Market Share Net Appliocations: http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0&qptimeframe=M&qpsp=138&qpnp=1. Last visited on 15 June 2010.

[Flat03] Flatscher Rony G.: “The Augsburg Version of BSF4ooRexx”. http://wi.wu-wien.ac.at/rgf/rexx/orx14/orx14_bsf4rexx-av.pdf. Last visited on 23 June.

[Flat04] Flatscher Rony G.: “Camouflaging Java as Object REXX”. http://wi.wu-wien.ac.at/rgf/rexx/orx15/2004_orx15_bsf-orx-layer.pdf. Last visited on 23 June.

[GeDC10] Mozilla Wiki: Gecko:DeCOMtamination: https://wiki.mozilla.org/Gecko:DeCOMtamination. Last visited on 26 March 2010.

[InXP01] XPCOM Part 1: An Introduction to XPCOM: http://www.ibm.com/developerworks/webservices/library/co-xpcom.html. Last visited on 26 March 2010.

[JBSF10] Jakarta BSF FAQ – What is Bean Scripting Framework: http://jakarta.apache.org/bsf/faq.html#what-is-bsf. Last visited on 23 June 2010.

[LaBi09] MDC – Language Bindings: https://developer.mozilla.org/en/XPCOM/Language_Bindings. Last visited on 19 April 2010.

[Mits10] Mitschek Andreas: Scripting Mozilla Applications with XPCOM and XUL. http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/BakkStuff/2010/201007_Mitschek/Mitschek_Bakkalaureatsarbeit_XPCOM_XUL-final-20100706.pdf. Last visited on 14 July 2010.

[MoPl10] MDC – Places: https://developer.mozilla.org/en/Places. Last visited on 14 June 2010.

[OORe09] Open Object Rexx – About Open Object Rexx: http://www.oorexx.org/about.html. Last visited on 23 June 2010.

[OvXP09] MDC - Overview of XPCOM: https://developer.mozilla.org/En/Creating_XPCOM_Components/An_Overview_of_XPCOM. Last visited on 26 March 2010.

[RMUF10] mozillaZine: Reducing memory usage – Firefox: http://kb.mozillazine.org/Memory_Leak. Last visited on 26 June

[SuXP01] XPCOM Part 3: Setting up XPCOM: http://www.ibm.com/developerworks/webservices/library/co-xpcom3.html. Last visited on 26 March 2010.

[XPCB01] XPCOM Part 2: XPCOM component basics: http://www.ibm.com/developerworks/webservices/library/co-xpcom2.html. Last visited on 26 March 2010.

List of Code Examples

Appendix

In the Appendix are listed all examples that were presented in the paper. Whereas in the paper only important fragments were used, here are the examples complete.

1. Example: WindowCreator.java

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsIAppStartup;
import org.mozilla.interfaces.nsIDOMWindow;
import org.mozilla.interfaces.nsIServiceManager;
import org.mozilla.interfaces.nsIWindowCreator;
import org.mozilla.interfaces.nsIWindowWatcher;

public class WindowCreator {
/* main() embeds XPCOM environment and opens an URL in a DOMWindow */

public static void main(String []args) throws Exception {

String targetURL = "http://www.wu.ac.at";
GREVersionRange[] range = new GREVersionRange[1];
range[0] = new GREVersionRange("1.8", true, "1.9+", true);
File grePath = null;
/* get a Mozilla class instance and the path to the Gecko Runtime
Environment (GRE) */
try {
grePath = Mozilla.getGREPathWithProperties(range, null);
}
catch (FileNotFoundException e) { }
if (grePath == null) {
System.out.println("found no GRE PATH");
return;
}
System.out.println("GRE PATH = " + grePath.getPath());
Mozilla mozilla = Mozilla.getInstance();
/** try embedding the XPCOM environment using the GRE path */
try {
mozilla.initialize(grePath);
mozilla.initXPCOM(grePath, null);
}
catch (IllegalArgumentException e) {
System.out.println("no javaxpcom.jar found in given path");
return;
}
catch (Throwable t) {
System.out.println("initXPCOM failed");
t.printStackTrace();
return;
}
/** XPCOM is successfully embedded */
System.out.println("\n--> initialized\n");
try {
```

```
// To get access to interfaces we get an instance of the XPCOM service
manager
nsIServiceManager serviceManager = mozilla.getServiceManager();
// Use contract ID (@mozilla.org/toolkit/app-startup;1) and IID to get
startup application nsIAppStartup
nsIAppStartup appStartup = (nsIAppStartup)serviceManager
      .getServiceByContractID("@mozilla.org/toolkit/app-startup;1",
      nsIAppStartup.NS_IAPPSTARTUP_IID);
// Get the nsIWindowCreator interface through appStartup
nsIWindowCreator windowCreator = (nsIWindowCreator)appStartup
      .queryInterface(nsIWindowCreator.NS_IWINDOWCREATOR_IID);
// Get the nsIWindowWatcher interface
nsIWindowWatcher windowWatcher = (nsIWindowWatcher)serviceManager
      .getServiceByContractID("@mozilla.org/embedcomp/window-watcher;1",
      nsIWindowWatcher.NS_IWINDOWWATCHER_IID);
// Set the window creator
windowWatcher.setWindowCreator(windowCreator);
// Create the DOMWindow with the supplied URL
nsIDOMWindow win = windowWatcher.openWindow(null, targetURL, "mywindow",
      "chrome,resizable,centerscreen", null);
// DOMWindow win is active window
windowWatcher.setActiveWindow(win);
// Start the XPCOM startup application
appStartup.run();
}
catch (XPCOMException e) { e.printStackTrace(); }
// shut down XPCOM embedding
mozilla.shutdownXPCOM(null);
}
}
```

2. Example: BookmarksManager.java

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsIServiceManager;
import org.mozilla.interfaces.nsINavBookmarksService;
import org.mozilla.interfaces.nsIIOService;
import org.mozilla.interfaces.nsIURI;

public class BookmarksManager {

public static void main(String []args) throws Exception {

GREVersionRange[] range = new GREVersionRange[1];
range[0] = new GREVersionRange("1.8", true, "1.9+", true);
File grePath = null;
/** get a Mozilla class instance and the path to the Gecko Runtime
Environment (GRE) */
try {
grePath = Mozilla.getGREPathWithProperties(range, null);
}
catch (FileNotFoundException e) { }
if (grePath == null) {
System.out.println("found no GRE PATH");
```

```
return;
}
System.out.println("GRE PATH = " + grePath.getPath());
LocationProvider locProvider = new LocationProvider(grePath);
Mozilla mozilla = Mozilla.getInstance();
/** try embedding the XPCOM environment using the GRE path */
try {
mozilla.initialize(grePath);
mozilla.initXPCOM(grePath, locProvider);
}
catch (IllegalArgumentException e) {
System.out.println("no javaxpcom.jar found in given path");
return;
}
catch (Throwable t) {
System.out.println("initXPCOM failed");
t.printStackTrace();
return;
}
/** XPCOM is successfully embedded */
System.out.println("\n--> initialized\n");
try {
// To get access to interfaces we get an instance of the XPCOM service
manager
nsIServiceManager serviceManager = mozilla.getServiceManager();
// Get the bookmarks service
nsINavBookmarksService bms = nsINavBookmarksService)serviceManager.
      getServiceByContractID("@mozilla.org/browser/nav-bookmarks-
      service;1", nsINavBookmarksService.NS_INAVBOOKMARKSSERVICE_IID);
// Get the IO service
nsIIOService ios = (nsIIOService)serviceManager.getServiceByContractID
      ("mozilla.org/network/io-service;1", nsIIOService.NS_IIOSERVICE_IID);
// Create a bookmark folder
long menuFolder = bms.getBookmarksMenuFolder();
long newFolderID = bms.createFolder(menuFolder, "search engine",
      bms.DEFAULT_INDEX);
// Create URI object
nsIURI uri = ios.newURI("http://google.com/", null, null);
// Check whether an URL is already bookmarked and add it to the bookmarks
if (bms.isBookmarked(uri)) {
System.out.println(uri.getPrePath() + " has been bookmarked");
} else {
bms.insertBookmark(newFolderID, uri, bms.DEFAULT_INDEX, "Google");
}
}
catch (XPCOMException e) { e.printStackTrace(); }
// shut down XPCOM embedding
mozilla.shutdownXPCOM(null);
System.out.println("finished");
}
}
```

3. Example: SaveToFile.java

```
import java.io.*;
import org.mozilla.xpcom.*;
import org.mozilla.interfaces.nsIAppStartup;
import org.mozilla.interfaces.nsIIOService;
import org.mozilla.interfaces.nsIServiceManager;
import org.mozilla.interfaces.nsIWebBrowserPersist;
import org.mozilla.interfaces.nsILocalFile;
import org.mozilla.interfaces.nsIURI;

public class SaveToFile {

public static void main(String []args) throws Exception {

GREVersionRange[] range = new GREVersionRange[1];
range[0] = new GREVersionRange("1.8", true, "1.9+", true);
File grePath = null;
try {
grePath = Mozilla.getGREPathWithProperties(range, null);
}
catch (FileNotFoundException e) { }
if (grePath == null) {
System.out.println("found no GRE PATH");
return;
}
System.out.println("GRE PATH = " + grePath.getPath());
LocationProvider locProvider = new LocationProvider(grePath);
Mozilla mozilla = Mozilla.getInstance();
/** try embedding the XPCOM environment using the GRE path */
try {
mozilla.initialize(grePath);
mozilla.initXPCOM(grePath, locProvider);
}
catch (IllegalArgumentException e) {
System.out.println("no javaxpcom.jar found in given path");
return;
}
catch (Throwable t) {
System.out.println("initXPCOM failed");
t.printStackTrace();
return;
}
/** XPCOM is successfully embedded */
System.out.println("\n--> initialized\n");
try {
// To get access to interfaces we get an instance of the XPCOM service
manager
nsIServiceManager serviceManager = mozilla.getServiceManager();
// create a persist
nsIWebBrowserPersist persist = (nsIWebBrowserPersist)
      serviceManager.getServiceByContractID
      ("@mozilla.org/embedding/browser/nsWebBrowserPersist;1",
      nsIWebBrowserPersist.NS_IWEBBROWSERPERSIST_IID );
// do the save
nsIIOService ios = (nsIIOService)serviceManager.getServiceByContractID
      ("mozilla.org/network/io-service;1", nsIIOService.NS_IIOSERVICE_IID);
nsILocalFile file = (nsILocalFile)serviceManager.getServiceByContractID
```

```
      ("@mozilla.org/file/local;1", nsILocalFile.NS_ILOCALFILE_IID);

nsIURI uri = ios.newURI("http://google.com/", null, null);
file.initWithPath("C:/newFile.html");
persist.saveURI(uri, null, null, null, "", file);
}
catch (XPCOMException e) { e.printStackTrace(); }
// shut down XPCOM embedding
mozilla.shutdownXPCOM(null);
System.out.println("finished");
}
}
```

4. Example: WindowCreator.rex

```
.bsf~bsf.import('java.io.File','File')
.bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
.bsf~bsf.import('org.mozilla.xpcom.GREVersionRange','GREVersionRange')
.bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
.bsf~bsf.import('org.mozilla.interfaces.nsIDOMWindow','nsIDOMWindow')
.bsf~bsf.import('org.mozilla.interfaces.nsIServiceManager','nsIServiceManager')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowCreator','nsIWindowCreator')
.bsf~bsf.import('org.mozilla.interfaces.nsIWindowWatcher','nsIWindowWatcher')
/* Target URL to open (explicit or by BSF bean) */
targetUrl = .bsf~bsf.lookupBean('targetUrl')
if targetUrl = .nil then targetUrl = 'http://www.wsj.com'
/* Insert Path to your XULRunner installation directory, which contains the
JavaXPCOM library "javaxpcom.jar" */
grePathName = "C:\Program Files\xulrunner"
/* Initiate XPCOM embedding */
path = .System~getProperty('GRE_PATH')
-- set grePathName manually (see first line)
if path = .nil then grePath = .File~new(grePathName)
else grePath = .File~new(path)
say 'Gecko Runtime Engine path: ' grePath~getPath
mozilla = .Mozilla~getInstance
mozilla~initialize(grePath)
mozilla~initXPCOM(grePath, .nil)
say 'Mozilla XPCOM initialized!'
/** Get the Service Manager (responsible for acquiring XPCOM objects) */
serviceManager = mozilla~getServiceManager
/** Retrieve necessary property values and XPCOM interface IIDs */
appStartupID = .nsIAppStartup~NS_IAPPSTARTUP_IID
windowCreatorID = .nsIWindowCreator~NS_IWINDOWCREATOR_IID
windowWatcherID = .bsf~bsf.getStaticValue(.nsIWindowWatcher,
     'NS_IWINDOWWATCHER_IID')
winProps = "width=1000, height=600, resizable, centerscreen,
     scrollbars='yes', status='yes'"
/** Set up the application and load the new window with interface
nsIWindowWatcher */
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)
window = windowWatcher~openWindow(.nil,targetUrl,'myWindow',winProps,.nil)
windowWatcher~setActiveWindow(window)
appStartup~run
/** Terminate XPCOM embedding */
mozilla~shutdownXPCOM(.nil)
say 'Mozilla XPCOM embedding finished!'
::requires BSF.cls -- adds BSF support to Java and ooRexx scripts
```

5. Example: CookieManager.rex

```
.bsf~bsf.import('java.io.File','File')
.bsf~bsf.import('java.lang.System','System')
.bsf~bsf.import('org.mozilla.xpcom.Mozilla','Mozilla')
.bsf~bsf.import('org.mozilla.xpcom.GREVersionRange','GREVersionRange')
.bsf~bsf.import('org.mozilla.interfaces.nsIAppStartup','nsIAppStartup')
.bsf~bsf.import('org.mozilla.interfaces.nsIDOMWindow','nsIDOMWindow')
.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.nsICookieManager','nsICookieManager')
.bsf~bsf.import('org.mozilla.interfaces.nsICookie','nsICookie')
/** Target URL to open (explicit or by BSF bean) */
targetUrl = .bsf~bsf.lookupBean('targetUrl')
if targetUrl = .nil then targetUrl = 'http://www.derstandard.at'
grePathName = "C:\Program Files\xulrunner"
/** Initiate XPCOM embedding */
path = .System~getProperty('GRE_PATH')
if path = .nil then grePath = .File~new(grePathName)
else grePath = .File~new(path)
say 'Gecko Runtime Engine path: ' grePath~getPath
mozilla = .Mozilla~getInstance
mozilla~initialize(grePath)
mozilla~initXPCOM(grePath, .nil)
say 'Mozilla XPCOM initialized!'
/* Get the Service Manager */
serviceManager = mozilla~getServiceManager
/* Retrieve necessary property values and XPCOM interface IIDs */
appStartupID = .bsf~bsf.getStaticValue(.nsIAppStartup,
    'NS_IAPPSTARTUP_IID')
windowCreatorID = .bsf~bsf.getStaticValue(.nsIWindowCreator,
    'NS_IWINDOWCREATOR_IID')
windowWatcherID = .bsf~bsf.getStaticValue(.nsIWindowWatcher,
    'NS_IWINDOWWATCHER_IID')
winProps = "width=1000, height=600, resizable, centerscreen,
    scrollbars='yes', status='yes'"
/* Set up the application and load the new window with interface
nsIWindowWatcher */
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)
window = windowWatcher~openWindow(.nil,targetUrl,'URLOpener',winProps,.nil)
```

```
windowWatcher~setActiveWindow(window)
appStartup~run
cookieManagerID = .bsf~bsf.getStaticValue(.nsICookieManager,
    'NS_ICOOKIEMANAGER_IID')
cookieManager = serviceManager~getServiceByContractID
     ('@mozilla.org/cookiemanager;1', cookieManagerID)
cookieID = .bsf~bsf.getStaticValue(.nsICookie, 'NS_ICOOKIE_IID')

iter = cookieManager~getEnumerator
do while iter~hasMoreElements
     cookies=iter~getNext
     cookie=cookies~queryInterface(cookieID)
     say cookie~getName
     say " "cookie~getHost
end

cookieManager~removeAll
iter2 = cookieManager~enumerator
if iter2~hasMoreElements then
say 'Cookies not deleted' else
say 'All cookies deleted'
/** Terminate XPCOM embedding */
mozilla~shutdownXPCOM(.nil)
say 'Mozilla XPCOM embedding finished!'
::requires BSF.cls -- adds BSF support to Java and ooRexx scripts
```

6. LocationProvider.java

(LocationProvider is an implementation of nsIAppFileLocProvider, which is used for XPCOM initialization.)

```
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;
}
}
```