

"Camouflaging Java as Object Rexx"

2004 International Rexx Symposium
Sindelfingen/Böblingen, Germany (May 2004)

Rony G. Flatscher (Rony.Flatscher@wu-wien.ac.at)
Wirtschaftsuniversität Wien, Austria (<http://www.wu-wien.ac.at>)

▼ Agenda

- BSF, BSF4Rexx
 - Architecture
 - Example
- Object Rexx wrapper "**BSF.cls**"
 - Overview
 - Classes and methods
 - Examples
- Roundup and Outlook

▼ BSF

- Bean Scripting Framework
 - A Java framework, making it easy for Java to invoke scripts in non-Java scripting languages
 - E.g. JavaScript, NetRexx
 - Originally developed by IBM as open source
 - Part of IBM's WebSphere to allow scripts to be deployed within Java Server Pages (JSP)
 - Fall 2003 handed over to jakarta.apache.org
 - Used e.g. in [ant](#), [xerces](#)

▼ BSF4Rexx

- BSF with a Rexx engine
 - Allows the usage of Rexx from BSF
 - Any Java program can invoke Rexx
 - Rexx scripts are able to communicate with Java objects, if made available by the Java program
 - Allows Java to be used as a huge Rexx function library
 - The public methods and public fields of every Java object and Java class object can be used by Rexx
 - If necessary, Java can be started up by Rexx

BSF4Rexx, Example Java using Rexx

```
import com.ibm.bsf.*; // BSF support
import java.io.*;    // exception handling
/** Java program which demonstrates how easy it is to invoke Rexx via BSF. */
public class TestSimpleExec
{
    /** Running an in-line defined Rexx script. */
    public static void main (String[] args) throws IOException
    {
        try
        {
            BSFManager mgr      = new BSFManager ();
            String      scriptCode = "SAY 'Rexx was here!'"; // a Rexx statement
            // invoke Rexx from Java via BSF
            mgr.exec("rex", "any debug info", 0, 0, scriptCode);
        }
        catch (BSFException e)
        {
            e.printStackTrace();
        }
    }
}
```

Yields:
Rexx was here!

BSF4Rexx, Example Rexx Using Java

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

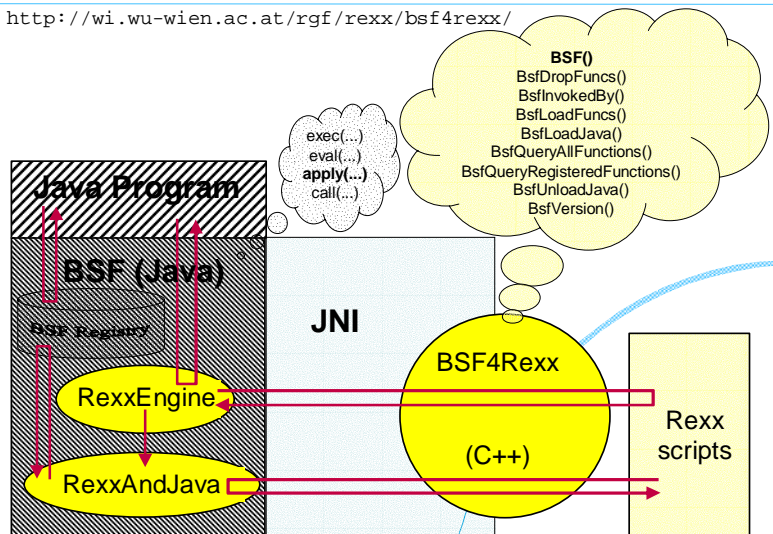
```
say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')
```

Yields, e.g.:

```
java.version: 1.4.2
```

BSF4Rexx Architecture

<http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/>



BSF4Rexx (BSFRegistry)

Pre-registered Java Objects

- 1) "Class.class"
- 2) "Object.class"
- 3) "Method.class"
- 4) "Array.class"
- 5) "String.class"
- 6) "System.class"
- 7) "Boolean.class"
- 8) "boolean.class"
- 9) "Byte.class"
- 10) "byte.class"
- 11) "Character.class"
- 12) "char.class"
- 13) "Double.class"
- 14) "double.class"
- 15) "Integer.class"
- 16) "int.class"
- 17) "Long.class"
- 18) "long.class"
- 19) "Float.class"
- 20) "float.class"
- 21) "Short.class"
- 22) "short.class"
- 23) "Void.class"
- 24) "void.class"

BSF4Rexx

BSF()-Subfunctions, 1

- (1) call BSF "addEventListener", beanName, eventSetName, filter, eventText
- (2) x=BSF("arrayAt", arrayObject, idx1 [, ...])
- (3) l=BSF("arrayLength", arrayObject)
- (4) call BSF "arrayPut", arrayObject, newValue, idx1 [, ...])
- (5) call BSF "arrayPutStrict", arrayObject, typeIndicator, newValue, idx1 [, ...])
- (6) a=BSF("createArray", componentType, capacity1 [, ...])
- (7) w=BSF("wrapArray", arrayObject)
- (8) res= BSF("exit" [, [retVal] [, time2wait in msec]])
- (9) v=BSF("getFieldValue", beanName, fieldName)
- (10) p=BSF("getPropertyValue", beanName, propertyName, index)
- (11) s=BSF("getStaticValue", className, fieldName)
- (12) res=BSF("invoke", beanName, methodName, arg1 [...])
- (13) res=BSF("invokeStrict", beanName, methodName, typeIndicator1, arg1 [, ..., ...])
- (14) o=BSF("lookupBean", beanName)
- (15) t=BSF("pollEventText" [, timeout in msec])
- (16) call BSF "postEventText", eventText, priority

BSF4Rexx

BSF()-Subfunctions, 2

- (17) o=BSF("registerBean", beanName, beanType, arg1 [...])
- (18) o=BSF("registerBeanStrict", beanName, beanType, typeIndicator1, arg1 [..., ...])
- (19) v=BSF("setFieldValue", beanName, fieldName, newValue)
- (20) v=BSF("setFieldValueStrict", beanName, fieldName, typeIndicator, newValue)
- (21) v=BSF("setPropertyValue", beanName, propertyName, index, newValue)
- (22) v=BSF("setPropertyValueStrict", beanName, propertyName, index, typeIndicator, newValue)
- (23) call BSF "setRexxNullString", newString
- (24) call BSF "sleep", time2sleep in msec
- (25) str=BSF("unregisterBean", beanName)
- (26) v=BSF("version")
- (27) e=BSF("wrapEnumeration", enumerableObject)

BSF4Rexx, Typing Issue, 1 "Strict"

- A newer version than the "Augsburg" version of BSF4Rexx
 - Beta version can be downloaded from
<http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/>
 - Allows to omit type information usually needed for Java
 - Java is a strongly typed programming language, Rexx is not!
 - "strict" allows to supply explicit type information
 - Needed under rare circumstances where Java methods of the same name and same number of arguments exist, but differ in the type of their arguments only

BSF4Rexx, Typing Issue, 2 "Strict"

- "Type indicators" precede the argument in BSF()-subfunctions containing the word "Strict"
- "Type indicators" are one of the following strings
 - **BO**olean, **B**yte, **C**har, **D**ouble, **F**loat, **I**nt, **L**ong, **O**bject, **S**hort, **S**tring
 - Only bold and uppercase letters need to be given
 - Java type information is given in the HTML documentation
 - "BOolean", "Byte", "Char", "Double", "Float", "Int", "Long", "SHort", "String" are the Java "primitive" data types
 - "Object" is *any* Java object

Camouflaging Java, 1

BSF.cls

- "BSF.cls"
 - An Object Rexx package
 - Defines routines, classes and methods which hide the procedural interface from Object Rexx programs
 - Wraps all BSF()-subfunctions into Object Rexx Methods
 - Allows to import Java classes explicitly into Object Rexx in the form of Object Rexx proxy classes
 - Allows to create Object Rexx proxy objects which interact with the appropriate Java objects

Camouflaging Java, 2

BSF.cls

- "BSF.cls"
 - Supports Java array objects as Object Rexx array proxies
 - Allows using Java array objects as if they were Object Rexx array objects
 - Hence indexing of proxy arrays starts with 1 (and not 0)!
 - Takes advantage of Object Rexx' **UNKNOWN** mechanism
 - Allows a rather simple implementation of the needed forwarding mechanism, which forwards Object Rexx messages to Java and causes the appropriate Java methods to be invoked
 - Among other things, takes advantage of the Object Rexx destructor mechanism to automatically free registered Java objects from the BSFRegistry
 - Hence, no orphaned objects in the BSFRegistry ("object leaks")

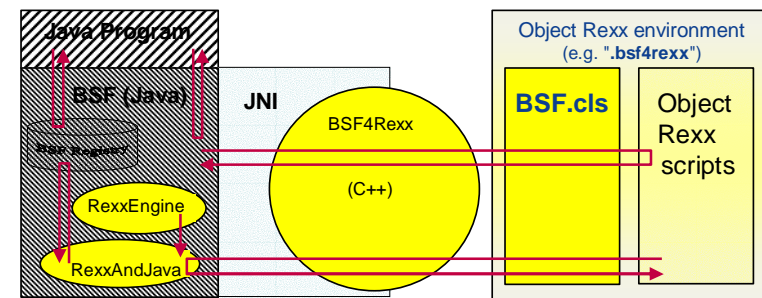
BSF4Rexx, Example Object Rexx Using Java

```
/* Object Rexx version */  
say "java.version:" .bsf4rexx-system.class ~getProperty('java.version')  
::requires "BSF.cls" -- loads the Object Rexx (camouflaging) support
```

Yields, e.g.:

```
java.version: 1.4.2
```

Camouflaging Java, 3 Architecture



BSF4Rexx (BSFRegistry)

Pre-registered Java Class Objects

- Object Rexx directory "**.BSF4Rexx**"

- | | |
|-------------------------------|-----------------------------|
| 1) .bsf4rexx~Class.class | 13) .bsf4rexx~Double.class |
| 2) .bsf4rexx~Object.class | 14) <i>.bsf4rexx~double</i> |
| 3) .bsf4rexx~Method.class | 15) .bsf4rexx~Integer.class |
| 4) .bsf4rexx~Array.class | 16) <i>.bsf4rexx~int</i> |
| 5) .bsf4rexx~String.class | 17) .bsf4rexx~Long.class |
| 6) .bsf4rexx~System.class | 18) <i>.bsf4rexx~long</i> |
| 7) .bsf4rexx~Boolean.class | 19) .bsf4rexx~Float.class |
| 8) <i>.bsf4rexx~boolean</i> | 20) <i>.bsf4rexx~float</i> |
| 9) .bsf4rexx~Byte.class | 21) .bsf4rexx~Short.class |
| 10) <i>.bsf4rexx~byte</i> | 22) <i>.bsf4rexx~short</i> |
| 11) .bsf4rexx~Character.class | 23) .bsf4rexx~Void.class |
| 12) <i>.bsf4rexx~char</i> | 24) <i>.bsf4rexx~void</i> |

BSF.cls, 1

Public Routines and Classes

- Routine **bsf.checkResult**
 - Expects a string, returns proxy object if a Java object, string else
- Class **BSF**
 - Proxy class to camouflage Java
- Class **BSF_PROXY**
 - Subclass of **BSF**
 - Wraps a string referring to a BSFRegistry entry into a BSF proxy object

BSF.cls, 2

Proxy Class **BSF**

- Execute "BSF.cls" either with **call** or **::requires**

```
call "BSF.cls"  
::requires "BSF.cls"
```

- Allows to import Java classes and interact with them as if they were Object Rexx classes

```
.bsf~import(rexxName, javaName)  
.bsf~import("javaFrame", "java.awt.Frame")  
f=.javaFrame~new("hi!")~show~toFront~setSize(200,100)
```

- Allows to create Java objects

```
.bsf~import("javaFrame", "java.awt.Frame")  
f1=.javaFrame~new("hi!") -- using an imported Java class  
f2=.BSF~new("java.awt.Frame", "hi!") - using .BSF directly
```

BSF.cls, 3

Proxy Class **BSF**

- Proxy objects

- Object Rexx objects which represent Java objects

- Such Java objects *must* be stored in the **BSFRegistry!**

- Sending Java messages to Object Rexx proxies will usually raise the **UNKNOWN** condition

- **UNKNOWN** method forwards the unknown message with the supplied arguments to Java
- Any resulting value will be returned to Object Rexx
 - If result is a Java object, then an Object Rexx proxy object will be returned

BSF.cls, 4

Proxy Class **BSF**

- Procedural BSF()-subfunctions available as (mangled) instance methods:
 - (1) bsf.addEventListener
 - (2) bsf.exit
 - (3) bsf.invoke
 - (4) *bsf.invoke**Strict***
 - (5) bsf.getFieldValue
 - (6) bsf.setFieldValue
 - (7) *bsf.setFieldValue**Strict***
 - (8) bsf.getPropertyValue
 - (9) bsf.setPropertyValue
 - (10) *bsf.setPropertyValue**Strict***
- Procedural BSF()-subfunctions available as class methods:
 - (11) exit
 - (12) sleep
 - (13) lookupBean
 - (14) pollEventText
 - (15) getStaticValue
 - (16) postEventText
 - (17) wrapArray
 - (18) createArray
 - (19) wrapEnumeration
 - (20) setRexxNullString

BSF.cls – Some Remarks, 1

Strong Typing

- Strong typing
 - If necessary, use the "strict" version of the methods
 - If needed while creating instances of Java classes (i.e. need for a strict version of the class method "new")
 - Import the appropriate Java class into Object Rexx
 - `.bsf~import("rexName", "javaName")`
 - Use "**newStrict**" instead of "**new**"
 - Importing will create a "**newStrict**" class method on the fly
 - `.bsf~import("javaFrame", "java.awt.Frame")`
 - `f1=javaFrame~newStrict("String", "Hi there!")`
 - `-- or:`
 - `f2=javaFrame~new("Hi there!")`

BSF.cls – Some Remarks, 2

Creating and Using Java Arrays

```
-- create a two-dimensional (5x10) Java Array of type String
arr=.bsf-createArray(.bsf4rexx-string.class, 5, 10)

arr[1,1]="First Element in Java array."      -- place an element
arr~put("Last Element in Java array.", 5, 10) -- place another one

do i over arr      -- loop over elements in array
  say i
end
```

Yields:

```
First Element in Java array.
Last Element in Java array.
```

Roundup and Outlook, 1

- Using the new beta version of "BSF4Rexx" there is no type information needed anymore
 - As a result it is *much easier to use* for Rexx programmers
- "BSF.cls" successfully camouflages Java as Object Rexx
 - Object Rexx proxy classes and proxy objects
 - Object Rexx messages are forwarded to the appropriate Java objects
 - Java arrays appear as if they were Object Rexx arrays

▼ Roundup and Outlook, 2

- BSF4Rexx
 - New version in the works, a beta available
 - An even newer version in alpha (on this machine ☺)
 - Will get a new namespace, namely
 - org.rexxla.bsf
 - Will be the same for IBM's and Apache's BSF
 - JNI-DLLs will cater for the differences
 - Java code remains the same for both implementations
 - Easier to maintain and to deploy
 - Glimpse on some other features
 - Will add multithreadability for Object Rexx
 - Additional features to simplify usage even more
 - Plan to distribute the final version via SourceForge