“CAMOUFLAGING JAVA AS OBJECT REXX”

Rony G. Flatscher

University of Augsburg, Germany

„The 2004 International Rexx Symposium“, Böblingen, Germany,

May 3rd - May 6th, 2004.

ABSTRACT

The Java runtime environment (JRE) is available for practical every operating system in the world and installed on most computers. The functionality of the Java classes that build the JRE has been constantly updated to reflect the advances in the software technology thereby making bleeding edge software concepts available to Java programmers. For that reason the JRE has been an attractive target for making its functionality available to Rexx programs in the form of external Rexx functions, notably with the "BSF4Rexx" (Bean Scripting Framework for Rexx) technology introduced at past International Rexx Symposiae. BSF4Rexx supplies a procedural interface to Java, such that Rexx programs need to simply use CALL-statements or function-calls to bridge into Java.

As Object Rexx is object-oriented an object-oriented interface to Java may be desirable as this may reduce the complexity to refer to Java. This article introduces and discusses the architecture and the implementation of Object Rexx wrapper classes that hide the procedural interfaces from Object Rexx programmers by embedding the procedural interfaces of BSF4Rexx in Object Rexx methods, allowing e.g. the invocation of Java methods transparently via Object Rexx messages.

Among other things it will be demonstrated, how Java objects are created and sent messages to interactively via a keyboard using the Rexx "rexxtry.rex" program in a command line interface.

Keywords: Object Rexx, Java, BSF4Rexx, UNKNOWN, proxy classes, proxy objects

1 INTRODUCTION

At the 2001 International Rexx symposium the first incarnation of the "BSF4Rexx" - the "Bean Scripting Framework for Rexx" - got introduced to the Rexx community [Flat01]. Taking advantage of the IBM open source project "Bean Scripting Framework" it has become possible to invoke Rexx programs from Java programs in an easy and straight-forward manner. In turn such invoked Rexx programs could call back into Java and take advantage of the wealth of the functionality implemented in Java classes and made available via Java objects.

At the 2003 International Rexx symposium a new, improved version of BSF4Rexx was introduced [Flat03], which allows Rexx programs to start Java and thereafter use it as a huge external Rexx function library.

This article builds on the"Augsburg version of BSF4Rexx" and introduces an Object Rexx module which allows Object REXX programmers to use Java classes and Java objects, as if they were Object REXX classes and Object REXX objects, effectively camouflaging Java.[1)]

[1)] As this article is based on [Flat03] the reader is advised to study that article first.

2 BSF - ARCHITECTURE AND APPLICATION

The "Bean Scripting Framework" (BSF) has begun its life as an IBM alphaworks project, which allows IBM employees to make their work available to the world. In the case that a particular alphaworks project [W3Alpha] draws the attention of other developers it may be the case that such a project turns into a so-called "developer work", which usually means that the project gets more attention and resources from IBM, but may also be employed in other IBM products like "IBM WebSphere" [W3WebSphere].

BSF defines and implements a Java framework which enables Java programmers to invoke programs from Java, which are written in a non-Java programming language like JavaScript or Perl. The initial support for non-Java programming languages by IBM in BSF 2.2 concentrated on Java implemented interpreters, like Mozilla's Rhino [W3Rhino] or Mike Cowlishaw's[2)] NetRexx [W3NetRexx] and in addition to the Microsoft ActiveX scripting languages JScript and VBScript, available on the Windows platform only.

In the fall of 2002 the entire open source project was handed over to the Jakarta project of the Apache organization and got released with the version number "BSF 2.3". This way the technology can be used in additional Apache funded/initiated open source projects like "`ant`" [W3Ant] or "`xerces`" [W3Xerces].

BSF4Rexx [Flat01, Flat03] is a software package to allow BSF to deploy Rexx and Object REXX as additional scripting languages for Java. In order to facilitate the interfacing to Java from invoked Rexx programs quite a comprehensive set of functions and subfunctions have been devised. As a result of this endeavor all the functionality of Java as implemented in its class hierarchies can be regarded as a huge external Rexx function library which can be directly used! As Java and its class libraries are available on all major operating system and hardware platforms, Rexx programs using the Java class library can be run unchanged on all those platforms!

2) Mike F. Cowlishaw is the original creator of the Rexx programming language and an IBM fellow.

Usually computer users have the „Java runtime environment“ (JRE) installed on their computers, even if they do not realize it. This is usually the case because WWW browsers like Netscape (Mozilla) or Opera depend on Java to execute Java applets that may be referred to or embedded in HTML documents that the user retrieved via the WWW. For that reason Rexx programmers using Java classes can expect their programs to be executable on all platforms.[3)]

2.1 A Brief Overview of the Augsburg Version of BSF4Rexx

This section briefly introduces the reader how one uses BSF and BSF4Rexx to allow a Java program to execute a Rexx program, how a Rexx program can invoke a method on a Java (class) object and finally gives an architectural overview of the Augsburg version of BSF4Rexx and the interfaces to Java made available by it.

2.1.1 Java Using Rexx

Figure 1 depicts a Java Programm which invokes the Rexx interpreter using BSF, which displays „Rexx was here!“. In order to invoke the Rexx interpreter and have it execute some Rexx code, the Java programmer nees to refer to the BSF package (line # 1), get a BSFManager (line # 11) and use it to invoke Rexx (line # 14). The Rexx code gets supplied as a single string, including all carriage returns and line feeds, if any.[4)]

2.1.2 Rexx using Java

Figure 2 depicts a Rexx program that retrieves and shows the version of the Java virtual machine it is using. If the Rexx program was invoked by Java, then the external Rexx function named „BSF()“ has been registered by Java already and the do block (lines # 4 through # 8) is not executed. On the other hand, if the main external Rexx function of the BSF4Rexx package needs to be loaded, it is clear that Java is not (yet) running.

3) It may be necessary, though, that on such client computers the path to the Java runtime library needs to be set explicitly. Cf. the documentation that accompanies BSF4Rexx to learn how to set up the environment correctly, allowing BSF4Rexx to become fully operational.

4) If the Rexx code is stored in an external file, then the Java programmer needs to read the content of that file into a String and supply it (line # 14).

```
/* 1 */  import com.ibm.bsf.*;   // BSF support
/* 2 */  import java.io.*;       // exception handling
/* 3 */  /** Java program which demonstrates how easy it is to invoke Rexx via BSF. */

/* 4 */  public class TestSimpleExec
/* 5 */  {
/* 6 */    /** Running an in-line defined Rexx script. */
/* 7 */    public static void main (String[] args) throws IOException
/* 8 */    {
/* 9 */        try
/* 10 */        {
/* 11 */          BSFManager mgr        = new BSFManager ();
/* 12 */          String     scriptCode = "SAY 'Rexx was here!'"; // a Rexx statement
/* 13 */                    // invoke Rexx from Java via BSF
/* 14 */          mgr.exec("rexx", "any debug info", 0, 0, scriptCode);
/* 15 */        }
/* 16 */        catch (BSFException e)
/* 17 */        {
/* 18 */          e.printStackTrace();
/* 19 */        }
/* 20 */    }
/* 21 */  }
```

Figure 1: Nutshell Example for a Java Program Invoking a Rexx Program Via BSF.

Therefore, after registering all the external BSF4Rexx external Rexx functions (lines # 5 through # 6 in figure 2) the Java virtual machine is loaded by Rexx and the Java-side of the BSF4Rexx support is initialized (line # 7).

In either case, line # 9 uses the external Rexx function „BSF()“ to interact with Java. Using the Java class „java.lang.System“ and its method („function“) named „getProperty“ one can retrieve the values of the defined Java properties. The single argument to „getProperty“ in this case is a string, hence the type indicator „STring“ is given before the argument value„java.version“. The returned value is a string containing the version number of the Java virtual machine, which Rexx then displays. If the installed

```
/* 1 */   /* classic Rexx version, querying the installed Java version */
/* 2 */   /* load the BSF4Rexx functions and start a JVM, if necessary */
/* 3 */ if rxFuncQuery("BSF") = 1 then   /* BSF() support not loaded yet ? */
/* 4 */ do
/* 5 */   call rxFuncAdd "BsfLoadFuncs", "BSF4Rexx", "BsfLoadFuncs"
/* 6 */   call BsfLoadFuncs /* registers all remaining BSF functions */
/* 7 */   call BsfLoadJava  /* loads Java */
/* 8 */ end
/* 9 */ say "java.version:" bsf('invoke','System.class','getProperty',‘STring’, -
                    'java.version')
```

Figure 2: Rexx Program Retrieving the Version of the Java Virtual Machine in Use.

Java virtual machine carries the version number „1.4.2“ then the output of the Rexx program in 2 would be: „java.version: 1.4.2“.

2.1.3 Architecture and Interfaces of the Augsburg Version of BSF4Rexx

Figure 3 depicts the overall architecture of the Augsburg version of BSF4Rexx.

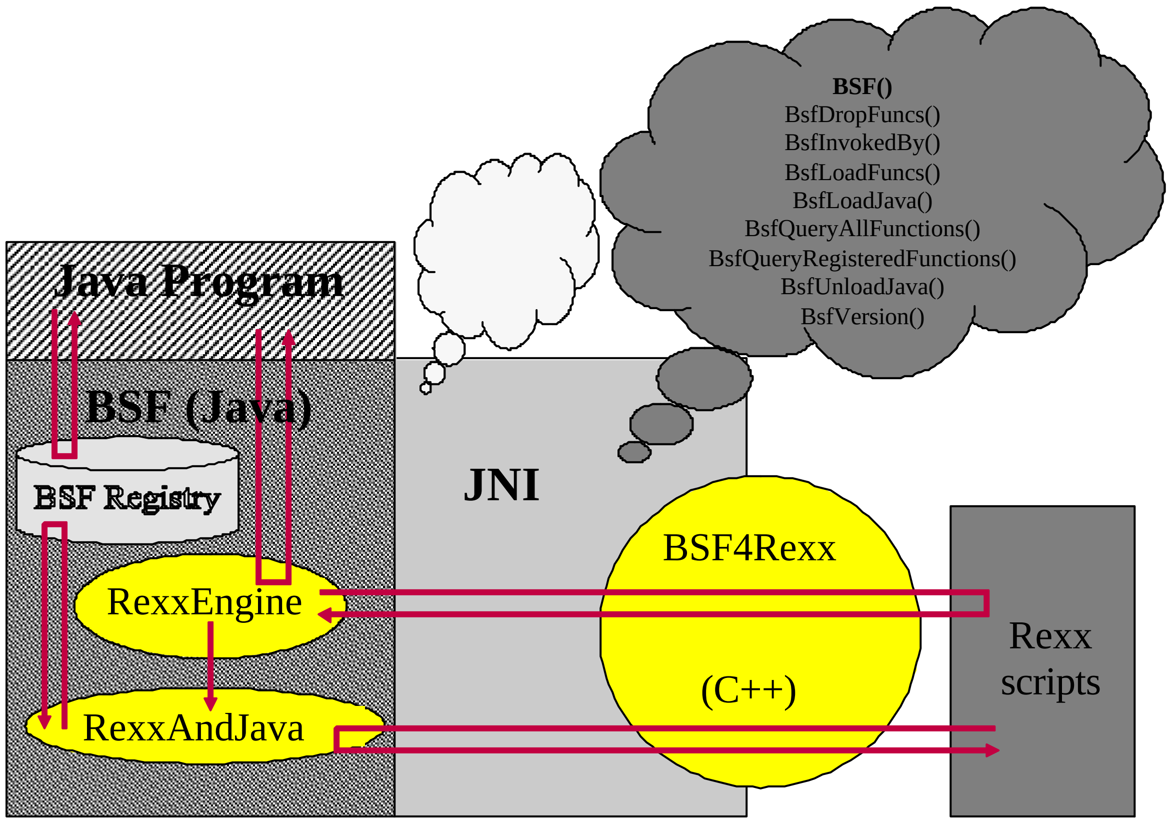

Figure 3: The Architecture of the Augsburg Version of BSF4Rexx.

The „Augsburg“ version [Flat03] builds upon the „Essener“ [Flat01] version of BSF4Rexx. Therefore it is still possible to invoke Rexx scripts from Java with the same interface. In addition it now is possible for Rexx to start the Java virtual machine (JVM), i.e. the Java runtime environment and then interface with Java (class) objects using the functionality of the Java classes as „external Rexx functions“!

In BSF each Java (class) object one creates and needs to interact with will be stored on the Java side in a („BSF“) registry which is comparable to an Object REXX directory. Knowing the string key value of a Java object in the BSF registry allows REXX

Java class object	**Name in BSF-Registry**	**Name in ".bsf4rexx" directory**
Fundamental Java class objects		
`java.lang.`**`Object`**	`Object.class`	`Object.class`
`java.lang.`**`Class`**	`Class.class`	`Class.class`
Additional, useful Java class objects		
`java.lang.reflect.`**`Array`**	`Array.class`	`Array.class`
`java.lang.`**`String`**	`String.class`	`String.class`
`java.lang.`**`System`**	`System.class`	`System.class`
Primitive Java datatypes (pseudo class objects) and their object-oriented Java class objects		
boolean*.class*	***boolean.class***	*boolean*
`java.lang.`**`Boolean`**	`Boolean.class`	`Boolean.class`
byte*.class*	***byte.class***	*byte*
`java.lang.`**`Byte`**	`Byte.class`	`Byte.class`
char*.class*	***char.class***	*char*
`java.lang.`**`Character`**	`Character.class`	`Character.class`
double*.class*	***double.class***	*double*
`java.lang.`**`Double`**	`Double.class`	`Double.class`
float*.class*	***float.class***	*float*
`java.lang.`**`Float`**	`Float.class`	`Float.class`
int*.class*	***int.class***	*int*
`java.lang.`**`Integer`**	`Integer.class`	`Integer.class`
long*.class*	***long.class***	*long*
`java.lang.`**`Long`**	`Long.class`	`Long.class`
short*.class*	***short.class***	*short*
`java.lang.`**`Short`**	`Short.class`	`Short.class`
void*.class*	***void.class***	*void*
`java.lang.`**`Void`**	`Void.class`	`Void.class`

Figure 4: List of Pre-registered Java (class) Objects in the BSF Registry.

programmers to denote the exact Java object as stored in the registry at the Java side. To ease the interfacing with Java there are a few Java (class) objects pre-registered in the BSF registry.

Knowing the string key values (case of letters is significant!) as depicted in the table in figure 4 (column # 2), allows a REXX programmer to indicate the Java class (type), whenever it is needed. One particularly important application area where this service comes in handy is when one wishes to create Java arrays, which always need to be of a given class (type).

2.2 Beyond the Augsburg Version of BSF4Rexx

At the time of this writing (2004) there have been a few improvements applied to the Augsburg version. The most notable one is the new ability to forgoe type indicators

Indicator	Datatype
"**bo**olean"	the value 0 (false) or 1 (true)
"**by**te"	a byte value
"**c**har"	a single (UTF-8) character
"**d**ouble"	a double value
"**f**loat"	a float value
"**i**nt"	an integer value
"**l**ong"	a long value
"**o**bject"	a Java object which is registered with the BSF registry (the immediately following argument is the string serving as the key for retrieving the desired Java object from the BSF registry).
"**sh**ort"	a short value
"**st**ring"	a string value (UTF-8)

Figure 5: Type Indicators („typeIndicator“ in Figure 6 Below) for Java Arguments.

(figure 5, only bold printed letters need to be given, case is not relevant), which in the Augsburg version had to preceed the individual arguments. Figure 6 depicts the external Rexx function „BSF()“ which allows Rexx to interact with Java, whereby the first argument indicates the „subfunction“ to be carried out at the Java side.

```
call BSF "addEventListener", beanName, eventSetName, filter, eventText
x=BSF("arrayAt", arrayObject, idx1 [, ...])
l=BSF("arrayLength", arrayObject)
call BSF "arrayPut", arrayObject, newValue, idx1 [, ...])
call BSF "arrayPutStrict", arrayObject, typeIndicator, newValue, idx1 [, ...])
a=BSF("createArray", componentType, capacity1 [, ...])
w=BSF("wrapArray", arrayObject)
res= BSF( "exit'" [, [retVal] [, time2wait in msec]])
v=BSF("getFieldValue", beanName, fieldName)
p=BSF("getPropertyValue", beanName, propertyName, index)
s=BSF("getStaticValue", className, fieldName)
res=BSF("invoke", beanName, methodName, arg1 [,...])
res=BSF("invokeStrict", beanName, methodName [, typeIndicator1, arg1]... )
o=BSF("lookupBean", beanName)
t=BSF("pollEventText" [, timeout in msecs])
call BSF "postEventText", eventText, priority
o=BSF("registerBean", beanName, beanType, arg1 [,...])
o=BSF("registerBeanStrict", beanName, beanType [, typeIndicator1, arg1]... )
v=BSF("setFieldValue", beanName, fieldName, newValue )
v=BSF("setFieldValueStrict", beanName, fieldName, typeIndicator, newValue)
v=BSF("setPropertyValue", beanName, propertyName, index, newValue )
v=BSF("setPropertyValueStrict", beanName, propertyName, index, typeIndicator, newValue)
call BSF "setRexxNullString", newString
call BSF "sleep", time2sleep in msecs
str=BSF("unregisterBean", beanName)
v=BSF("version")
e=BSF("wrapEnumeration", enumerableObject)
```

Figure 6: Rexx External Function „BSF()“ with its Subfunctions.

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

Figure 7: Rexx Program of Figure 2 without a Type Indicator.

Although it is not necessary anymore to supply type indicators for subfunctions there are some very rare situations where no guessing/inferring of the correct signature is possible. Therefore it is important to still supply the ability to Rexx programmers to uniquely indicate the types of the arguments using the type indicator strings of figure 5 above. The BSF-subfunctions honoring type indicators contain the string „Strict" in the name of their subfunctions in figure 6 above, e.g. „invokeStrict".

3 CAMOUFLAGING JAVA

Having an object-oriented version of Rexx it may prove useful to create an abstraction layer which hides the external Rexx function interface from Object REXX programmers and instead make it possible to interact with Java objects with the object-oriented means Object REXX supplies. If such a layer can be designed and implemented that its results makes Java objects and Java class objects appear to be Object Rexx objects, then a true camouflaging will take place.

The current BSF4Rexx package [W3BSF4REXX] contains an Object REXX file named „BSF.cls", the „Bean Scripting Framework" supporting module. This module defines Object REXX routines, classes and methods which hide the procedural interface of the external Rexx function package („BSF4Rexx.dll", resp. „libBSF4Rexx.so"). It „reconstructs" an object-oriented interface to the object-oriented Java runtime environment.

All procedural BSF-subfunctions as depicted in figure 6 above are either explicitly or implicitly wrapped up in Object REXX methods. To ease the interaction with Java for Object REXX programmers it is possible to import Java classes into Object REXX and address them thereafter as if they were genuine Object REXX classes, helping to camouflage Java.5) Such Object REXX classes that represent Java classes are also called „proxy classes" in this article, instances of them are called „proxy objects" to indicate that they are representing Java objects. They allow for sending Object REXX messages to the proxy objects which will eventually dispatch the corresponding Java messages to the Java object.6)

Another service supplied via the module „BSF.cls" is turning Java array objects into proxy Object REXX array objects. One is therefore able to interact with such proxies as if they were Object REXX array objects, using them in DO...OVER loops, and indexing them with the base „1" instead of the Java base of „0", making Java array objects

5) The „import" idea was inspired by the OS/2 implementation of Object REXX where it has been possible to import SOM and DSOM classes into Object REXX that may have been implemented in any of the SOM/DSOM supported computer languages.

6) This effect is easily created by taking advantage of Object REXX' UNKNOWN mechanism.

```
/* Object Rexx version */
say "java.version:" .bsf4rexx~system.class ~getProperty('java.version')

::requires BSF.cls -- loads the Object Rexx (camouflaging) support
```

Figure 8: Object REXX Program Retrieving the Version of the Java Virtual Machine in Use.

almost indistinguishible from Object REXX array objects. As a result Object REXX programmers do not have to be aware of the intricacies the Java syntax sometimes implies.

Another important implication of applying the proxy concept to camouflage Java as Object Rexx is the taking advantage of the Object REXX garbage collector for purposes of maintaining the registry of the BSF4Rexx Java sided support.

Everytime a Java object is registered with the BSF registry on the Java side that object remains in the registry until it is explicitly deleted from it. In order to do so, BSF4Rexx programs need to invoke the BSF-subfunction „unregisterBean“ supplying the string key value. After the reference counter for the Java objects in the registry drops to zero, will that Java object be removed from the registry and made available for being garbage collected by the Java garbage collector. If a programmer forgets about unregistering, than such Java objects remain in the registry forever, even if they are never used again. Clearly, such a mistake should be avoided in order to save resources. If using the „BSF.cls“ module everytime an Object REXX proxy gets garbage collected on the Object REXX side, the destructor („UNINIT“) method is run which will unregister the corresponding Java object from the BSF registry. This mechanism runs automatically and therefore it is impossible that Java objects remain in the BSF registry forever.

Figure 8 depicts an Object REXX program that retrieves the Java version of the Java runtime environment in use. Comparing this program to the classic REXX one in figure 2 above it is evident, that the Object REXX version is much shorter. Also, if used to the object-oriented syntax it can be seen that it is easy to interact with Java proxies, because one can interact with them as if they were Object REXX objects. The requires directive causes the interpreter to invoke „BSF.cls“ before the first statement of the program is executed. That required call will set up the camouflaging infrastructure for

Object REXX and also will automatically start up Java, in case that it is not running yet. In addition the initialization code of „BSF.cls“ will create a direcotry object named „BSF4Rexx“ and stores it in the local environment for easy access by any Object REXX program.[7] „.BSF4REXX“ will contain entries which refer directly to all of the pre-registered Java class objects of the BSF registry as shown in figure 4. Therefore „.bsf4rexx~system.class“ will return the proxy object for the Java class „java.lang.System“ which possesses a (static) method named „getProperty“ which is used to query the version of Java. Again, one merely needs to send an ObjectREXX message to the proxy object and supply the argument in round parenthesis.

Figure 9 depicts the architecture put in place to camouflage Java as Object REXX. This camouflaging in effect takes place through the Object REXX module „BSF.cls“ which either needs to get called by the programmer before accessing Java or which needs to be required using the Object REXX requires directive, which causes it to be executed before the first statement of the program is executed. This way it is guaranteed, that the BSF4Rexx Object REXX environment has been set up before the first statement (at the very top) of the Object REXX program get executed. Or with other words: any such program can rely on the BSF4Rexx environment being available when it runs.

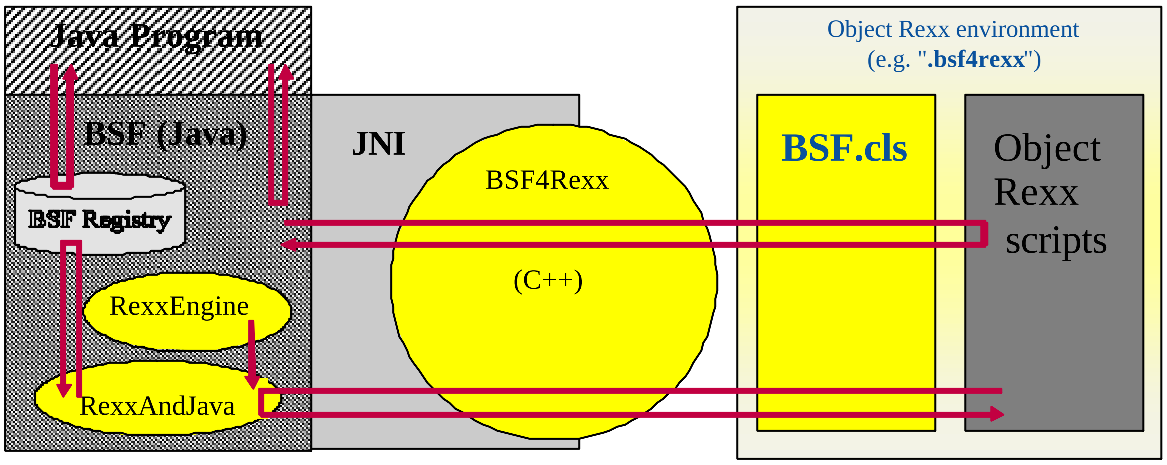

Figure 9: Architecture to Camouflage Java as Object REXX (Using „BSF.cls“).

[7] Any entry in the Object Rexx environments named .local or .environment can also be directly addressed by prepending the name of the entry with a dot (.), e.g. the entry „BSF4REXX“ can be addressed with „.BSF4REXX“ (note the dot at the beginning of the name, making this an „environment symbol“).

3.1 Referring to the Preregistered Java Class Objects

As mentioned above, BSF4Rexx will pre-register the most important Java class objects in the BSF registry by the names as documented in figure 4 above. These string values are case sensitive and therefore the case of the string key is significant. This poses a few problems in the case where the primitive Java datatype string keys are named the same as their Java class counterparts (also dubbed „reference datatypes“). The reason being, that in REXX every statement is translated into uppercase before it gets executed.[8] Therefore a statement like „.bsf4rexx~short.class“ and „.bsf4rexx~Short.class“[9] will both get uppercased to the same string „.BSF4REXX~SHORT.CLASS“, so no distinction is possible after the translation took place.

To solve this particular problem the following convention was chosen: the index (a string) for the Object REXX directory object „.BSF4Rexx“ will not contain the trailing string „.class“ for primitive datatypes. Figure 10[10] shows how one can get the proxy class objects for the respective Java class objects from the Object REXX „.BSF4Rexx“ directory. The Java class proxies for the primitive datatypes are shown in italic.

```
.bsf4rexx ~ Class.class              .bsf4rexx~Double.class
.bsf4rexx ~ Object.class             .bsf4rexx ~ double
.bsf4rexx ~ Method.class             .bsf4rexx~Integer.class
.bsf4rexx ~ Array.class              .bsf4rexx ~ int
.bsf4rexx ~ String.class             .bsf4rexx~Long.class
.bsf4rexx ~ System.class             .bsf4rexx ~ long
.bsf4rexx~Boolean.class              .bsf4rexx~Float.class
.bsf4rexx ~ boolean                  .bsf4rexx ~ float
.bsf4rexx~Byte.class                 .bsf4rexx~Short.class
.bsf4rexx ~ byte                     .bsf4rexx ~ short
.bsf4rexx~Character.class            .bsf4rexx~Void.class
.bsf4rexx ~ char                     .bsf4rexx ~ void
```

Figure 10: Getting the Java Class Object Proxies from the BSF4Rexx Directory Object.

[8] This pre-execution step involves also the removal of duplicate blanks between the tokens. Strings, enquoted with single or double quotes, are left intact.

[9] Please note the capital „S“ in the string „Short.class“.

[10] It can be seen from figure 10 that the message operator (~) may be enclosed by blanks in Object REXX. Using those blanks for formatting should make it a little easier to focus on the messages to retrieve the primitive datatype proxy class objects.

3.2 Public Routine and Public Classes in „BSF.cls“

The Object REXX module „BSF.cls“ defines a couple of classes and routines, of which the following are made available publically:

- Class „BSF“: this is the Object Rexx proxy class,
- Class „BSF_PROXY“: this allows to create proxy objects for already registered Java objects,
- Routine „bsf.checkResult“: this either returns a string or a proxy object for a Java object or a proxy array object for a Java array object.

3.2.1 The Public Routine „bsf.checkResult“

This routine expects one argument only, a string that denotes the „beanName“, ie. the string value serving as the key into the BSF registry on the Java side. If this argument is „.nil“ (the Object REXX object representing „null“) or the string does not contain the character „@“[11] then the received argument is returned.

If from the supplied string it can be inferred that indeed a Java array object is referenced, this routine will return an appropriate Object Rexx proxy object, which can be interacted with as if it was an Object Rexx array. This includes the usage of the messages „AT“, „PUT“, „[]“, „[]=“, „DIMENSION“, „ITEMS“, „MAKEARRAY“, and „SUPPLIER“. Otherwise a simple Object Rexx proxy object is returned, allowing to interact with the Java object, that gets represented by it.

This public routine is usually only used by the BSF module itself.

3.2.2 The Public Object REXX Class „BSF“

The class „.BSF“ is the main class defined in the module „BSF.cls“. It implements the forwarding mechanism such that messages sent to its instances will get dispatched on the Java side via the BSF4Rexx infrastructure. For this purpose the UNKNOWN-mechanism gets employed: if a method by the name „UNKNOWN“ exists

[11] The string must be formed according to the Java convention: fully qualified name of the class, „@“, and some (hexadecimal) number. In the case of a Java array object the number of dimensions are indicated with the according number of opening square brackets „[“.

Instance Methods of Class „.BSF“	**Class Methods of Class „.BSF“**
`bsf.addEventListener`	`exit`
`bsf.exit`	`sleep`
`bsf.invoke`	`lookupBean`
bsf.invokeStrict	`pollEventText`
`bsf.getFieldValue`	`getStaticValue`
`bsf.setFieldValue`	`postEventText`
bsf.setFieldValueStrict	`wrapArray`
`bsf.getPropertyValue`	`createArray`
`bsf.setPropertyValue`	`wrapEnumeration`
bsf.setPropertyValueStrict	`setRexxNullString`

Figure 11: BSF()-Subfunctions Available as Methods to the Proxy Class „.BSF“.

in an Object REXX class, then the Object REXX runtime system will invoke it in the case that a method cannot be found that matches the name of the received message. The UNKNOWN method will receive two arguments: the name of the unknown message, and .nil or an array object containing all arguments that were sent together with the message. With this information in hand it becomes possible at runtime to determine the sought for Java method (i.e. the name of the unknown message) and using the BSF4Rexx function „BSF()“, subfunction „invoke“ one can invoke the appropriate Java method.

In addition most of the important „BSF()“-subfunctions are made available in the form of methods of the class „.BSF“, the instance methods starting with the string „BSF.“, the class methods without it. This way it is fairly simple to e.g. add an eventAdapter to a Java object, by sending the Object REXX proxy object the message „bsf.addEventAdapter“ with the appropriate arguments. Figure 11 documents these methods.

```
/* 1 */ f=.BSF~new("java.awt.Frame", "hi!")
/* 2 */ f~~show ~~toFront ~~setSize(200,100)

/* 3 */ ::requires BSF.cls -- loads the Object Rexx (camouflaging) support
```

Figure 12: Creating a Java Object and Interacting with It via Object REXX.

Figure 12[12)] depicts an example in which the class „.BSF“ is used to create an instance of the Java class „java.awt.Frame“, defining as its title bar the string „hi!“. This Java object is then shown on the user interface, brought forward and its size gets changed to a width of 200 pixels and height of 100 pixels.

The creation of the Java object in figure 12 will be referenced counted on the Java side by the supporting BSF4Rexx class („RexxAndJava.java“). Once this Object REXX program ends all destructors for all of the Object REXX objects will run, invoked by the Object REXX garbage collector. The BSF class defines a destructor („UNINIT“) that will use the BSF()-subfunction „unregisterBean“ to automatically decrease the reference counter on the Java side.

Importing Java Classes into Object REXX. One interesting feature of the class „.BSF“ is its class method „IMPORT“ which allows importing Java classes into Object Rexx and regard them as Object REXX classes instead. The first argument is the class name for Object Rexx, the second argument is the fully qualified Java class name. Importing a Java class into Object Rexx not only allows using the „NEW“ class method of the Object REXX proxy class, but will also add a „NEWSTRICT“ class method, which allows to indicate the Java types for each argument directed at the Java class constructor(s).

```
/* 1 */ .BSF~import("javaFrame", "java.awt.Frame")
/* 2 */ f=.javaFrame~new"hi!")
/* 3 */ f~~show ~~toFront ~~setSize(200,100)

/* 4 */ ::requires BSF.cls -- loads the Object Rexx (camouflaging) support
```

Figure 13: Importing a Java Class into Object REXX.

```
/* 1 */ .BSF~import("javaFrame", "java.awt.Frame")
/* 2 */ f=.javaFrame~newStrict("String", "hi!")
/* 3 */ f~~show ~~toFront ~~setSize(200,100)

/* 4 */ ::requires BSF.cls -- loads the Object Rexx (camouflaging) support
```

Figure 14: Importing a Java Class into Object REXX, Using „NEWSTRICT“.

12) The program will run unchanged on any operating system and hardware, because it uses Java classes to create a GUI window. The example also employs cascading messages in line # 2, as the three messages are all directed at the Java frame object referred to via variable „f“.

Figure 13 demonstrates the usage of the „IMPORT“ class method of class „.BSF“, it matches the semantics of figure 12 above. Figure 14 shows how one can use the „NEWSTRICT“ method to create an instance of the Java frame class. In the case of the Java class „java.awt.Frame“ the „NEWSTRICT“ method is not necessary as the signature of the available constructors make it trivial to determine the correct constructor.

Creating Java arrays and interacting with them from Object REXX. Figure 15 depicts an Object REXX program that uses the class method „createArray“ of the class „.BSF“ to create a Java array object of type „String“, having a maximum number of entries in the first dimension of five and in the second dimension of 10, allowing a total of 50 string values to be stored in that Java array object. To indicate to Java the mandatory type of the array object one needs to supply a Java class object. Because of the „BSF.cls“ module all Java class objects that are preregistered in the BSF registry on the Java side can be referred to via the Object Rexx directory object „.BSF4REXX“.

The Object Rexx code demonstrates that the array proxy object indeed can be dealt with as if it was an Object Rexx array object:

- square brackets (invoking the method „[]=“) to indicate the dimensions and the value to assign to location [1,1],

```
/* 1 */ -- create a two-dimensional (5x10) Java Array of type String
/* 2 */ arr=.bsf~createArray(.bsf4rexx~string.class, 5, 10)
/* 3 */
/* 4 */ arr[1,1]="First Element in Java array."       -- place an element
/* 5 */ arr~put("Last Element in Java array.", 5, 10) -- place another one
/* 6 */
/* 7 */ do i over arr      -- loop over elements in array
/* 8 */    say i
/* 9 */ end

/* 10 */ requires BSF.cls -- loads the Object Rexx (camouflaging) support
```

Figure 15: Creating a Java Array and Referring to it via an Object Rexx Array Proxy Object.

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

Figure 16: Output of Program in Figure 15.

- a „PUT“ message following the very same syntax as Object Rexx arrays, first the value and then the respective indices,
- a „do ...over“ loop that loops over all Java objects stored in the Java array object. As can be seen from figure 16 empty elements of the Java array are not processed, as is the case with Object REXX arrays being used in a „do...over“ loop.

3.2.3 The Public Object REXX Class „BSF_PROXY“

This class allows turning a string key (formed as pointed out in footnote 11 above) denoting a Java object in the BSF registry into an Object Rexx proxy. This public class is usually only used by the BSF module itself.

4 SUMMARY AND OUTLOOK

This article introduced the reader to the camouflaging of Java as Object REXX by the means of an Object REXX module, „BSF.cls“, that itself employs the BSF4Rexx procedural interface via the external Rexx function BSF(). It also hints at additions, changes applied to the Augsburg version of BSF4Rexx, that alleviate the REXX programmer of supplying type information for each argument meant for Java. This draws from the next version of BSF4Rexx, where the latest beta will be always available from [W3BSF4REXX].

The services implemented in the module „BSF.cls“ and made available via the Object REXX class „.BSF“ allows for:

- using Object Rexx objects („proxy objects“) to interact with corresponding Java objects,
- Object REXX messages sent to proxy objects will be dispatched on the appropriate Java object on the Java side,
- Java array objects appear as if they were Object REXX array objects.

Future planned changes and enhancements of BSF4Rexx are:

- changing the Java namespace to „org.rexxla.bsf.engines.rexx“ for both, the IBM and the Apache version of BSF,
- adding support for Object REXX multithreaded execution of objects, allowing Object REXX threads to dispatch Java methods concurrently,
- possibly „boxing“of the primitive Java datatypes boolean, byte, short, int, long, float and double.

5 REFERENCES

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

[Flat96a] Flatscher R.G.: "Local Environment and Scopes in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat96b] Flatscher R.G.: "Object Classes, Meta Classes and Method Resolution in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

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

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

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

[W3Alpha] Homepage of the open source Apache project "ant", URL (2004-05-01): `http://www.alphaworks.ibm.com/`

[W3Ant] Homepage of the open source Apache project "ant", URL (2004-05-01): `http://ant.apache.org/`

[W3Apa] Homepage of the open source Apache organization, URL (2004-05-01): `http://www.apache.org/`

[W3BSF] Homepage of Apaches “Bean Scripting Framework” (BSF), URL (2004-05-01): `http://jakarta.apache.org/bsf`

[W3BSF4REXX] Beta test and release candidate site of the “BSF4Rexx” package, URL(2004-05-01): `http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/`

[W3Jakarta] Homepage of the open source Apache organization, URL (2004-05-01): `http://jakarta.apache.org/`

[W3Java] Java homepage, URL (2004-05-01): `http://java.sun.com/`

[W3JavaDoc] Java documentation homepage, URL (2004-05-01):
`http://java.sun.com/docs`/

[W3NetRexx] NetRexx homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2004-05-01): `http://www2.hursley.ibm.com/netrexx/`

[W3ObjRexx] Object Rexx homepage of IBM, URL (2004-05-01):
`http://www.ibm.com/software/ad/obj-rexx/`

[W3Rexx] Rexx homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2004-05-01): `http://www2.hursley.ibm.com/rexx/`

[W3RexxLA] Rexx homepage of the “Rexx Language Association”, URL (2004-05-01):
`http://www.RexxLA.org`

[W3RxTrans] Homepage of Mark Hessling's "RexxTrans", URL (2004-05-01):
`http://rexxtrans.sourceforge.net/index.html`

[W3Rhino] Rhino homepage, URL (2004-05-01): `http://www.mozilla.org/rhino`

[W3WebSphere] Homepage of IBM's WebSphere product, URL (2004-05-01):
`http://www.ibm.com/software/info1/websphere/index.jsp`

[W3Xerces] Homepage of the open source Apache project "Xerces", URL (2004-05-01): `http://xml.apache.org/xerces2-j/index.html`

Date of Article: 2004-05-01.

Published in: Proceedings of the „2004 International Rexx Symposium“, Böblingen, Germany, May 3nd - May 6th, 2004, The Rexx Language Association, Raleigh N.C., 2004.

Presented at: „2004 International Rexx Symposium“, Böblingen, Germany, May 4th, 2004.