“JAVA BEAN SCRIPTING WITH REXX”

Rony G. Flatscher
Department of Management Information Systems at the
Vienna University of Economics and Business Administration (Austria)
„12th International Rexx Symposium“, Raleigh, North Carolina, USA,
April 30th - May 2nd, 2001.

ABSTRACT

IBM has developed an open source and free software system called “Bean Scripting Framework” (BSF) which allows Java programs to easily invoke scripts and programs written in another language than Java. This article gives a bird eyes view of BSF by introducing its architecture and hints at the potential and ramifications of this interesting technology. It then concentrates on describing and documenting the implementation of a “Rexx engine”, which allows Rexx interpreters and Object Rexx interpreters to be incorporated into the BSF framework. This allows for employing Rexx and Object Rexx everywhere where IBM’s BSF is employed, as well as individually create Java programs which invoke and co-operate with Rexx and Object Rexx programs.

1 INTRODUCTION

IBM has been actively supporting the Java movement from the beginning by supporting the developing of numerous technologies in this area and actively seeking co-operation with other companies and organizations since the midlle of the 90ies of the 20th century. Within IBM developers have been getting the opportunity to make their work publically available via their alphaworks Web site [W3Alpha]. Some of these alphaworks projects are terminated, others may be developed further and some may be turned into IBM maintained open sourced and freely available developer works projects.

The “Bean Scripting Framework” (BSF) has been turned from an alphaworks to a developer works project at the beginning of 2001 and can be retrieved in its latest version 2.2 from [W3BSF]. This project has been dealing with the creation of interfaces from java to scripting languages, either implemented in Java themselves like NetRexx [W3NetRexx] from Mike F. Cowlishaw or Rhino [W3Rhino] from Netscape’s Mozilla project, or in non-Java languages like Perl or BML. The BSF package is written in pure Java and allows Java programs to call any supporting scripting language with or without passing arguments to supplied scripts in those languages.

Depending on the invocation it is possible for Java programs to retrieve return values from those scripts. Figure 1 depicts the four different types of invocating scripts from Java via the infrastructure made available by BSF:

- “exec”, supplies a script, no arguments and does not expect a return result from the scripting engine, which executes the script,
- “eval”, supplies a script (usually a single line of code), no arguments and expects a return result from the scripting engine, which executes the script,
- “apply”, supplies a multi-line script representing an anonymous function, optional arguments and expects a return result from the scripting engine, which executes the script,

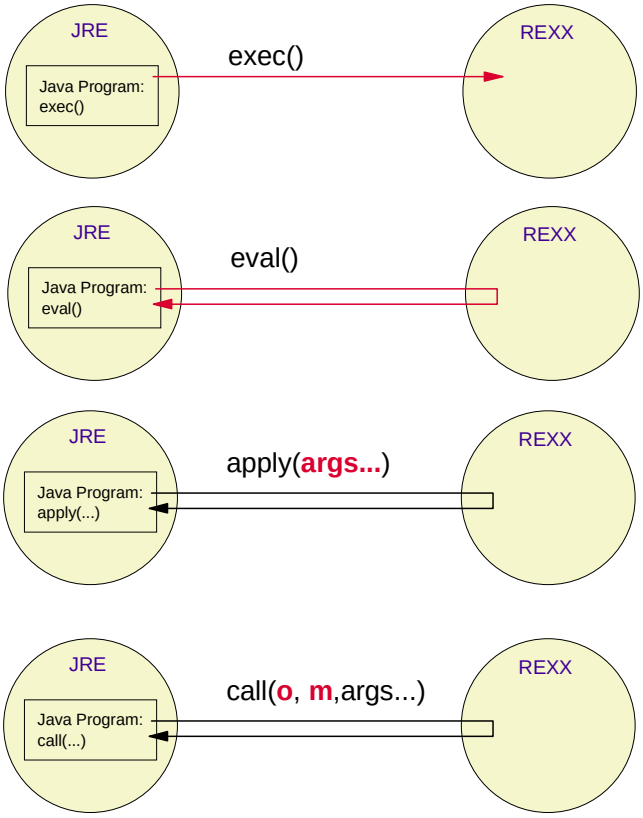

Figure 1: BSF - Invoking a Script (Example: Rexx Engine).

- “call”, supplies the target object, the name of the method (function) to invoke, supplies optionally arguments and expects a return result from the scripting engine, which executes the script.

The BSF package is accompanied with a simple Java program[1)] which allows for invoking any script from the command line with switches that determine whether the invocation occurs with the “eval” or the “exec” method.

The BSF package supplies important and very helpful services for use by the called scripts which ultimately allow those scripts to invoke methods on Java objects by calling back into Java via BSF.[2)] Figure 2 depicts the overall architecture of BSF, allowing for invoking scripts and supporting these by allowing to easily call back into Java to invoke methods on Java objects as well. In order to take advantage of BSF

1) This program - `com.ibm.bsf.Main.java` - also serves as an example of how easy it is to invoke *any* BSF engine from Java programs.

2) E.g. BSF supports the dynamic Java method resolution depending on the datatypes supplied with the name of a method which should get invoked on the given Java object.

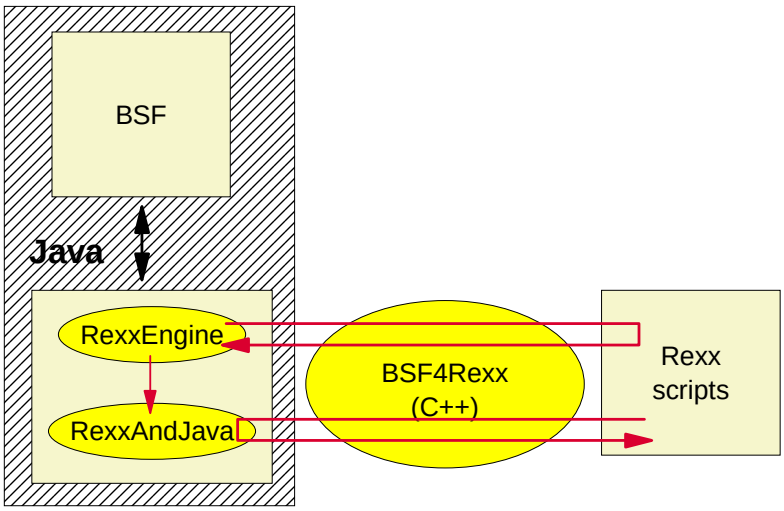

Figure 2: Interfacing (Object) Rexx with IBM's "Bean Scripting Framework" (BSF).

one needs to create a BSF "Engine" which is able to invoke the interpreter which executes the script passed from Java to Rexx.

Originally, IBM made the BSF package available via its Alphaworks site [W3Alpha], and starting with January 2001 changed its status to a supported open source project via their developer works site [W3BSF], which at the end of May 2001 is up to version 2.2. It supports the following languages out of the box: Java Script, Lotus' XSLT, NetRexx, Rhino, Visual Basic. This paper describes the interfaces made available with the implementation of a Rexx Engine for BSF.[3)]

[3)] Rexx scripts are able to refer to Java objects registered by BSF which are dubbed "beans" in the BSF documentation. The term "bean" in the BSF context denotes *any* Java object being stored in the BSF registry on the Java side and is not to be mixed up with the term "Java beans" as defined by Sun.

2 THE REXX BSF ENGINE

In the winter semester of 2000/2001 a student, Peter Kalender of the University of Essen (Germany), took over the assignment of a seminar project about evaluating IBM's Bean Scripting Framework and to venture possibilities to allow Rexx interpreters to be supported by it. By the middle of December 2000 he was able to present a basic implementation allowing Java programs to invoke Rexx scripts and making it possible for those Rexx programs to call back into Java and invoke methods on Java objects. By February 2001 he was able, after studying and learning Object Rexx, to come up with an Object Rexx wrapper class, allowing Object Rexx programs to use the messaging mechanism to address Java objects and have methods invoked upon them.[4)]

As the student had addressed many different and time consuming challenges (learning Rexx, learning BSF, creating C++ and Java support programs) one could not expect that he was able to create a full implementation of all aspects of BSF and of Java, allowing Rexx programmers to use Java to capacity, simply due to time restrictions. Yet, the basic evaluation and assessment as well as a prototypic implementation paved the way for creating a full implementation with taking advantage of these preliminary results, helping to dramatically cut down the needed time to get to all information necessary to achieve this task.

This section will document the resulting functionality made available in the full implementation of the Rexx BSF Engine which is aimed at making available all aspects of Java to Rexx programs. It concludes with a section describing the new version of the wrapper class for allowing Object Rexx to transparently create and access Java objects.

2.1 The Rexx Function Interface

Rexx programs being invoked via BSF are always invoked by a Java program. In the process of this invocation process the Java part ("RexxEngine.java") of the Rexx Engine makes sure that an external Rexx function, named "BSF" is registered and thereby made automatically available to the called Rexx program. This is done by

[4)] The seminar paper and the initial Object Rexx wrapper class can be found in [Kal01].

loading a dynamic link library written in C++, “BSF4Rexx”, which needs to be compiled for the target operating system and target hardware. The communication between Rexx and Java is been done via this DLL using the “Java Native Interface” (JNI) technology for being able to interact with Java using C++. The communication with Rexx on the Java side is realized with the Java program “RexxAndJava.java” with as much functionality being moved from the original C++ interface to the Java side, like the list of events pending to be read (polled) by the Rexx side.[5)]

The syntax rules in this article show optional arguments enclosed into square brackets. If an argument or a closing bracket is followed by three dots (“ellipses”), the argument or the bracketed arguments may be repeated multiple times, if necessary. In cases where the Java side needs to indicate “null” the Rexx string for indicating this is returned.[6)]

2.1.1 Passing Arguments Between Rexx and Java

The Java programming language is a strongly typed language, whith a compiler making sure that the declared datatypes are correctly used. Choosing of methods with the same name is done via the signature[7)] of such methods. By contrast, Rexx is not a strongly typed language and there is no notion of datatype per se. All Rexx values are strings which might get interpreted as numbers by context, e.g. if one wishes to add the content of two variables, which implies that they need to contain valid numbers.

Because of this fundamental difference, it is important for Rexx to indicate to Java, as which datatype a Rexx string is to be interpreted. It is this definition which gets used on the Java side to convert to the correct (indicated) datatype and to find a matching method, i.e. a method which arguments occur in the order indicated by Rexx and being of the indicated data type. Java’s primitive datatypes “boolean”, ”byte”, ”char”, “short”, ”int”, “long”, ”float” and ”double”, as well as Java’s datatype “String” are directly translated from and to Rexx strings. Java objects on the other

5) Therefore, should the need arise, that more Java functionality should be made available to Rexx, then “RexxAndJava.java” could be used to add that functionality for Rexx.

6) By default the value for representing Java’s “null” is the Rexx case sensitive string “.NIL” (note the leading point).

7) The “signature” consists of the return data type and the data types of the arguments of a method.

Indicator	Datatype
"**Bo**olean"	the value 0 (false) or 1 (true)
"**By**te"	a byte value
"**C**har"	a single (UTF8) 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 must be registered already with the BSF registry
"**Sh**ort"	a short value
"**St**ring"	a string value (UTF8)

Table 1: Datatype Indicator Strings.

hand are stored in a registry on the Java side with a unique name. Therefore, Java objects can be addressed and referred to by Rexx by name. In the case that an argument to a Java method refers to such a registered Java object, Rexx must indicate this explicitly by using the datatype “Object”. ,Table 1 shows the Java datatypes, with the Rexx strings denominating them.[8) 9)]

2.1.2 The Core Functions for Interfacing Rexx with BSF

This section describes the core functions enabling Rexx to interface with Java using IBM’s Bean Scripting Framework via the external Rexx function BSF(). These functions allow to create, free, lookup Java objects in the BSF repository, to invoke arbitrary methods on these objects (like function calls in Rexx, just on Java objects) and to terminate the Java Virtual Machine before Rexx exits.

8) Note the bold emphasize in the column “Indicator” in table 1 indicates the minimum (case-independent) letters one needs to supply to uniquely designate the Java datatype.

9) The Java datatypes “char” and “String” (consisting of char elements) are encoded in Unicode UTF-8. Therefore it may be necessary to translate the characters and strings from/to UTF-8, if using non-US ASCII characters.

2.1.2.1 Creating and Freeing Java Objects from Rexx

Rexx programs are able to create instances of Java classes, “Java objects”, and interact with these by invoking the Java methods available to those Java objects. In the context of the Bean Scripting Framework this process is in effect a sequence of two-steps: first the Java object is created on behalf of Rexx and secondly, that Java object is stored at the Java side in a registry and can be referred to merely by name. This in turns allows Rexx to refer to Java objects by name only by using the BSF interface. As long as the registered Java object is stored in the BSF registry it is available to Rexx. In the case that Java objects are no longer needed by Rexx, it should get removed from the BSF registry, in order for allowing Java to garbage collect unused Java objects and reclaim the resources they got reserved.

Figure 3 shows the syntax of creating an instance of a Java class which gets stored in the BSF registry. “beanName” can be a name supplied by Rexx which gets used to register the newly created Java object (dubbed “bean”) or can be left empty in which case a unique name is created on the Java side.[10)] “beanType” denominates the Java class which gets instantiated. In the case that it is necessary to supply arguments for creating a particular Java object, these arguments need to be given thereafter, each preceeded by the Java datatype as mentioned above. This function will return the name by which the created Java object got registered in the BSF registry and which must be used to refer to that object from Rexx.

```
res=BSF(‘registerBean’, beanName, beanType [, typeIndicator, arg]...)
```

Figure 3: Syntax for Creating and Registering a Java Object from Rexx.

Figure 4 shows the syntax of de-registering (freeing) of Java objects from the BSF registry. This call variant supplies the “beanName” by which the Java object got registered on the Java side by BSF. It will return the “beanName” which just got unregistered. In the case that the indicated bean does not exist in the registry, the

---

[10)] The Java side creates a unique name with the following algorithm as documented in the Java Software Development Kit, 1.3.0, for class “Object” and method “toString()” with the following Java code:

```
o.getClass().getName() + "@" + Integer.toHexString(o.hashCode())
```

Rexx string for indicating “null” gets returned.[11)]

```
res=BSF(‘unregisterBean’, beanName)
```

Figure 4: Syntax for Freeing a Java Object from Rexx.

2.1.2.2 Looking Up Registered and Declared Java Objects from Rexx

The Bean Scripting Framework allows Java programs and other scripting languages to store Java objects in the BSF registry independently of Rexx.[12)] If the name of such a registered Java object is known, it can be referred to and looked up by Rexx. The syntax of looking up a declared bean is given in figure 5. “beanName” indicates the name of the registered Java object. This function call returns the Rexx string for indicating “null”, if the Java object is not found, else the name of the reference to the Java object, which should get used in subsequenct referrals.[13)]

```
res=BSF(‘lookupBean’, beanName)
```

Figure 5: Syntax for Looking Up a Registered Java Object from Rexx.

2.1.2.3 Invoking Java Methods from Rexx

Figure 6 shows the syntax diagram for invoking methods[14)] on Java objects.[15)] “beanName” denominates the Java object stored by this name in the BSF registry on

---

11) This implementation uses reference counters on the Rexx created Java objects and effectively deletes the entry in the BSF registry only, if the reference counter drops to 0.

12) Cf. [W3BSF] using the online documentation on the methods “declare()” and “undeclare()”.

13) At present the implementation will reference count the access of a Java object with this function, if necessary even creating a new access name according to the rules layed out above for creating new Java objects via the “registerBean” function. Hence, should a looked up Java object not being needed anymore, one needs to free it explicitly via a “unregisterBean” call as layed out above.

14) “Invoking methods on objects” is comparable to calling a function on the data represented by the Java objects in classic Rexx. Hence the Rexx function call “REVERSE(‘abc’)” could be re-formulated in an OO-way by “’abc’~REVERSE”, meaning “invoke the method REVERSE on the string object ‘abc’”.

15) To find out which methods one can invoke on Java objects, consult the documentation of the respective Java classes the Java object was created from. The Java classes of the Java runtime environment (JRE) are documented in the “Java Development Kit” (JDK) of Sun ([W3Java]) or in the Java development kits of other companies like IBM.

the Java side. “methodName” indicates the name of the *public* method which should get invoked (“called”) on the object. Depending on the method one may need to supply arguments which follow thereafter in pairs of “typeIndicator” (cf. table 1 above) and “arg” (argument value). At the Java side the name of the method combined with the sequence and data types of the arguments is used to identify at runtime the method, which should get invoked.

This function returns the value the invoked method produced or the Rexx string representation of “null”. If a Java object is returned, then at the Java side it gets automatically registered, if needed, and the name to reference it will be returned. Such references to Java objects created as a result of a Java method invocation should get freed from the Rexx side with “unregisterBean” (cf. above) in order to allow the Java object to be garbage collected and to reclaim its reserved resources.[16)]

```
res=BSF(‘invoke', beanName, methodName [, typeIndicator, arg]...)
```

Figure 6: Invoking a Method on a Java Object from Rexx.

2.1.2.4 Terminating the Java Virtual Machine from Rexx

For Rexx scripts that are merely started from Java in order to get access and to use the Java runtime environment as a huge Rexx function package, it is necessary upon finishing execution to terminate the Java virtual machine which originally started the Rexx script. Figure 7 denotes the syntax of the BSF-call to exit the Java virtual machine by issuing a “System.exit(return_value)” at the Java side.

“return_value” allows to set the return value used at the Java side, which defaults to 0. “time2sleep” is a milli-second value to wait at the Java side before exiting the Java virtual machine which defaults to 100 ms. The delayed exiting of the Java virtual machine allows Rexx to cleanly terminate, i.e. closing and releasing the resources it used.[17)]

---

16) The Object Rexx wrapper class “BSF.cls” discussed further down will automatically unregister those Java objects, which are referred to by Object Rexx proxy objects which get garbage collected by Object Rexx.

17) Exiting the Java virtual machine implies immediate abortion of Rexx as well.

This function returns the verbatim value “SHUTDOWN, REXX !” as a reminder of the purpose of this function.

```
res=BSF(‘exit’ [, return value [, time2sleep]])
```

Figure 7: Syntax for Exiting the Java Virtual Machine from Rexx.

2.1.3 Auxiliary Functions for Interfacing Rexx with BSF

If writing Rexx programs to interact with Java objects, it is sometimes necessary or merely convenient to be able to use functions to achieve certain operations, which even might be realizable via a sequence of Java method invocations. This subsection introduces and documents the functions available to Rexx.

2.1.3.1 Retrieving a Public Field Value of a Java Object from Rexx

In Java it is common practice to access and change values of fields stored with objects, which are declared to be *public*. Figure 8 shows the syntax of the appropriate BSF call. Should a field have no value, the Rexx string representation for “null” is returned, otherwise the stored value. “beanName” refers to the Java object in the BSF registry, “fieldName” is the case sensitive name of a public field defined for the object in question.

```
res=BSF(‘getFieldValue’, beanName, fieldName)
```

Figure 8: Syntax for Retrieving the Value of a Public Java Object Field from Rexx.

2.1.3.2 Setting a Public Field Value of a Java Object from Rexx

In Java it is common practice to access and change values of fields stored with objects, which are declared to be *public*. Figure 9 shows the syntax of the appropriate BSF call. “beanName” refers to the Java object in the BSF registry, “fieldName” is the case sensitive name of a public field defined for the object in question, “argType” is a Rexx string according to table 1 above and “newValue” contains the value the new field is to be set to. In the case that no value should be stored with the public field in question, then the Rexx string representation for Java “null” has to be used which removes any values from the field.

```
res=BSF('setFieldValue', beanName, fieldName, argType, newValue)
```

Figure 9: Setting the Value of a Public Java Object Field from Rexx.

2.1.3.3 Retrieving a Property Value of a Java Object from Rexx

In Java some classes, denoted as “Java Beans”[18] may possess properties which may get queried. Figure 10 shows the syntax of the appropriate BSF call. Should a property have no value the Rexx string representation for “null” is returned, otherwise the stored value. “beanName” refers to the Java object in the BSF registry, “fieldName” is the case sensitive name of a public property defined for the object in question. As properties may be indexed an optional index can be given.

```
res=BSF('getPropertyValue', beanName, fieldName [, index])
```

Figure 10: Retrieving the Value of a Public Java Object Property from Rexx.

2.1.3.4 Setting a Property Value of a Java Object from Rexx

In Java it is common practice to access and change values of fields stored with objects, which are declared to be *public*. Figure 11 shows the syntax of the appropriate BSF call. “beanName” refers to the Java object in the BSF registry, “fieldName” is the case sensitive name of a public field defined for the object in question, “index” is optional in the case a property is not indexed and otherwise indicates the index of the property to set, “argType” is a Rexx string according to table 1 above and “newValue” contains the value the new field is to be set to. In the case that no value should be stored with the public field in question, then the Rexx string representation for Java “null” has to be used which removes any values from the property.

```
res=BSF('setPropertyValue', beanName, fieldName, [index], argType, newValue)
```

Figure 11: Setting the Value of a Public Java Object Property from Rexx.

---

[18] “Java Beans” as defined in the JDK must not be confused with the term used in IBM’s Bean Scripting Framework (BSF), which uses the term “bean” as well, but primarily denotes Java objects stored in the registry. It *may* be the case that BSF was originally developed for Java Beans and thereafter extended to interface with normal Java objects as well.

2.1.3.5 Retrieving a Static Value

Usually it is possible with the “getFieldValue” function to retrieve the value of a static public field as well.[19] However, there are situations, where this may not be (easily) possible, e.g. static public field with predefined values serving as constants. Such constants can be found with (uninstantiable) Java Interfaces or (uninstantiable) abstract Java classes. Figure 12 denotes the syntax of the call which allows Rexx to easily get at the appropriate values of public static fields. Should a field have no value the Rexx string representation for “null” is returned, otherwise the stored value. “className” refers to the Java class *or* interface and “staticFieldName” is the case sensitive name of the static public field which value should get retrieved.

```
res=BSF('getStaticValue', className, staticFieldName)
```

Figure 12: Retrieving the Value of a Public Static Java Object Field from Rexx.

2.1.3.6 Setting the Rexx String Representing Java’s “null”

Java uses a special constant (“null”) to indicate whether a field has no value or whether a method returns no valid object. Therefore it is very important to be able in Rexx to detect and to supply this important indicator. By default the *bsf4rexx* [W3BSF4R] implementation uses the case sensitive string “.NIL” to represent a Java “null”. Should the unlikely case arise, that some program uses this awkward string, then it becomes necessary for Rexx to be able to switch to a totally different string representation. Figure 13 denotes the syntax of the BSF-call which allows to set the string representing “null” on the Java side to the argument “newString”. This setting takes effect immediately.

```
res=BSF('setRexxNullString', newString)
```

Figure 13: Setting the Rexx String Representing Java’s “null” from Rexx.

[19] In Java an instance of a class (a Java object) is able to directly access its instance fields as well as the static (class) fields of the class it belongs to.

2.1.3.7 Retrieving an Element of a Primitive[20] Java Array

In the present implementation of *bsf4rexx* [W3BSF4R] the primitive Java arrays are supported from one to five dimensions. Figure 14 shows the syntax for retrieving an element of an array given the appropriate indices. “arrayObject” is any registered Java object representing a primitive Java array. *Note:* Java arrays start with an index of 0.[21]

```
res=BSF('arrayAt', arrayObject, i1 [, i2 [, i3 [, i4 [, i5]]]] )
```

Figure 14: Retrieving an Element of a Primitive Java Array Object from Rexx.

2.1.3.8 Retrieving the Length of a Primitive Java Array

Figure 15 shows the syntax for retrieving the length (number of elements) of a primitive Java array. “arrayObject” is any registered Java object representing a primitive Java array.

```
res=BSF('arrayLength', arrayObject )
```

Figure 15: Retrieving the Length (Number of Elements) of a Primitive Java Array Object from Rexx.

2.1.3.9 Setting a Value of a Primitive Java Array

In the present implementation of *bsf4rexx* [W3BSF4R] the primitive Java arrays are supported from one to five dimensions. Figure 16 shows the syntax for setting an element of an array given the appropriate type of the new value according to table 1 above, the new value itself and the indices determining the position in the array. “arrayObject” is any registered Java bean representing a primitive Java array.

---

[20] “Primitive” in this context indicates that these are Java arrays, which are not specializations of the Java collection classes. Such “primitive” Java arrays must be indexed directly in Java using square brackets.

[21] By contrast, Object Rexx arrays start with an index of one. The Object Rexx wrapper for BSF takes this into account and allows Object Rexx programs to index Java arrays as if they were Object Rexx array, transparently translating the index values if calling into Java.

Note: Java arrays start with an index of 0.

```
res=BSF('arrayPut', arrayObject, type, newValue, i1 [,i2[,i3[,i4[,i5]]]])
```

Figure 16: Retrieving an Element of a Primitive Java Array Object from Rexx.

2.1.3.10 Adding an Event Listener

For some Java classes it is possible for adding event listeners, which allow for indicating which event group, which event of this group should cause a pre-determined event text to be posted to Rexx. This can be very handy for many problems, e.g. for creating GUI applications with Java's "AWT" ("Abstract Window Toolkit") classes, where one is able to indicate which specific events should be signalled to Rexx by posting the specified event text. Figure 17 depicts the syntax for adding an event listener to a Java class which supports event notification.

"beanName" denotes the object to which an event listener is to be added, "eventSetName" defines the group of events, "filter" if not empty denotes the specific event which should cause "eventText" to be posted[22]. Such eventText can be retrieved with the BSF-function "pollEventText" and could be Rexx code which gets interpreted at runtime, but also any other text which then needs to get retrieved and analyzed by Rexx.

```
res=BSF('addEventListener', beanName, eventSetName, filter, eventText)
```

Figure 17: Adding an Event Listener to a Java Object Supporting it from Rexx.

2.1.3.11 Polling Events ("EventText"s) from Rexx

Rexx programs are able to retrieve information ("eventText") posted by event listeners or by Rexx programs as a result of the occurrence of an event. If there is no "eventText" pending, then this function waits until an "eventText" gets posted and returns it to the Rexx program. Figure 18 denotes the syntax for polling "eventText" from the list of "eventText"s maintained by the Java side. Sometimes it may be

[22] "eventText" posted by event listeners are put into the "normal" priority list of event texts to be dispatched. See description of the BSF functions "pollEventText" and "postEventText" below for more information on the priorities of event texts.

desirable to stop waiting on the occurrence of an “eventText” after a given time, so this function allows to define an optional time-out value in milliseconds. Should the polling of “eventText”s be timed-out the Rexx representation string of “null” will get returned.

In this *bsf4rexx* [W3BSF4R] implementation there are three priority levels available to place “eventText”s: “alarm”, “normal” and “batch”. In the case of mulithreaded Rexx programs it becomes therefore possible to post “eventText”s at the “alarm” priority level, which makes sure that they get retrieved before those at the “normal” and before those at the “batch” level, as the dispatching order follows exactly these priority levels from “alarm” to “batch”.

```
res=BSF('pollEventText' [, timeout])
```

Figure 18: Polling an Event Text from the List of Event Texts.

2.1.3.12 Posting an Event (“EventText”) from Rexx

Posting of event texts may be important for some applications, therefore this implementation makes posting of event texts available to Rexx as well. Figure 19 shows the syntax for posting event texts. “eventText” may be *any* Rexx string, which the Rexx program is able to retrieve via the function pollEventText() above, and which then may act accordingly. Sometimes it may even make sense to post an event text which represents executable Rexx code to be interpreted at runtime for calling desired functions or routines or in the case of Object Rexx to invoke certain methods on Object Rexx objects later on, when the text gets polled.

By default event text is posted at the “normal” priority level (value=1), which can be altered to the “alarm” priority level (value=2) or to the “batch” priority level (value=0). This allows for prioritizing the dispatching of pending event texts.

```
res=BSF('postEventText', eventText [, priority])
```

Figure 19: Posting an Event Text to the List of Event Texts.

2.1.4 Example of a Rexx Program Interfacing with Java

Figure 20 depicts a Rexx program which uses Java's runtime environment classes of the "abstract windows toolkit" (awt) to create a simple and portable GUI interface with the help of Java. Figure 21 shows the command line to invoke the program, if it is saved in a file called "ShowCount.rex", figure 22 shows an example snapshot of the running program.

```
/* "ShowCount.rex" - a Rexx program to count number of button presses  */
call BSF 'registerBean', 'Window',     'java.awt.Frame', 'String', 'Show count'
call BSF 'addEventListener', 'Window', 'window', 'windowClosing', 'call BSF
"exit"'

call BSF 'registerBean', 'Button',     'java.awt.Button', 'String', 'Press me!'
call BSF 'addEventListener', 'Button', 'action', '', 'call ShowSize'

call BSF 'registerBean', 'Label', 'java.awt.Label'
call BSF 'invoke', 'Label', 'setAlignment', 'I', '1'

call BSF 'invoke', 'Window', 'add', 'String', 'Center', 'Object', 'Label'
call BSF 'invoke', 'Window', 'add', 'String', 'South',  'Object', 'Button'
call BSF 'invoke', 'Window', 'pack'
call BSF 'invoke', 'Window', 'show'
call BSF 'invoke', 'Window', 'toFront'

i=0             /* set counter to 0     */

do forever
   a = bsf("pollEventText")     /* wait for an eventText to be sent     */
   interpret a                  /* execute as a Rexx program    */
   if result= "SHUTDOWN, REXX !" then leave /* JVM will be shutdown in 0.1sec */
end

exit

        /* show the actual number of times, you pressed the button      */
ShowSize:
   i=i+1
   call BSF 'invoke', 'Label', 'setText', 'String', "Press #" i
   return
```

Figure 20: Rexx Program which Uses Java's AWT for a GUI Interface.

```
java com.ibm.bsf.Main -mode exec -lang rexx -in ShowCount.rex
```

Figure 21: Invoking the Above Rexx Program via Java.

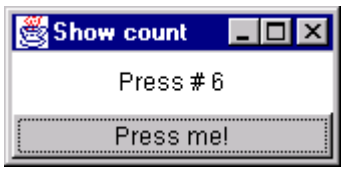

Figure 22: Rexx Program "ShowCount.rex" Using Java's AWT.

Both, the frame window and the button get an event listener attached to, which post the defined event text to the Rexx program, which polls and interprets it in the forever-loop. If the frame window gets closed the event text calling the exit-function is interpreted and the program ended. The only other event text posted and processed is the one defined with the event listener for the button, which causes the Rexx procedure “ShowSize” to be called every time the button is pressed. The procedure “ShowSize” increases the counter and sets the text for the label in order for the user to learn about the count of button presses.

2.2 The Object Rexx Wrapper Class “BSF.cls”

This section describes the support added to the *bsf4rexx* [W3BSF4R] package for users of the Object Rexx language, which being backwardly compatible with classic Rexx is able to use all of the Rexx interface functions described so far. The definitions in the file “BSF.cls” create an object-oriented wrapper for allowing transparently accessing the Java objects registered with BSF. Figure 23 depicts the Object Rexx program version of the classic Rexx program of figure 20 above. Invocation and presentation of the window remains exactly the same, if the Object Rexx program is saved with the file name “ShowRexx.rex”.

```
/* "ShowCount.rex" - an Object Rexx program to count number of button presses
*/
.bsf~import("awtFrame",  "java.awt.Frame")
.bsf~import("awtButton", "java.awt.Button")
.bsf~import("awtLabel",  "java.awt.Label")

win=.awtFrame~new("string", "Show Count")
win~bsf.addEventListener('window', 'windowClosing', '.bsf~exit')

but=.awtButton~new("string", "Press me!")
but~bsf.addEventListener('action', '', 'call ShowSize')

lab=.awtLabel~new ~~setAlignment('I', 1)

win ~~add("string", "Center", "Object", lab) ~~add("str", "South", "Obj", but)
win ~~pack ~~show ~~toFront

i=0             /* set counter to 0     */

do forever
   a = bsf("pollEventText")     /* wait for an eventText to be sent     */
   interpret a                  /* execute as a Rexx program    */
   if result= "SHUTDOWN, REXX !" then leave /* JVM will be shutdown in 0.1sec */
end

exit

        /* show the actual number of times, you pressed the button      */
ShowSize:
   i=i+1
   lab~setText('str', "Press #" i)
   return

::requires "BSF.cls"    -- get access to the Object Rexx support enhancement
```

Figure 23: Object Rexx Program which Uses Java’s AWT for a GUI Interface.

From this little example the following principles guiding the creation of “BSF.cls” become apparent:

- Java classes may get imported into the Object Rexx environment and can thereafter be used as if they were Object Rexx classes,

- creating instances of the imported Java classes creates Object Rexx objects serving as proxies for their Java counterparts,
- sending Object Rexx messages (terms right to the message operator “twiddle”, which is the character tilde) to these proxy objects invokes the methods on the Java side automatically.

In addition any returned BSF reference to a Java object registered in the BSF registry, causes a new Object Rexx proxy object to be created for reference. Should an Object Rexx proxy object not be used anymore and garbage corrected on the Object Rexx side, then the cleanup code of the destructor method “UNINIT” of class “BSF” unregisters the Java object from the BSF registry automatically.

Should the Java side return a primitive Java array with no more than five dimensions, then an array-like Object Rexx proxy is created, allowing to treat that array as if it was an Object Rexx array. In the case of a PUT-operation one needs to prepend the new value with the Rexx datatype indicator string as documented in table 1 above, such that the BSF interface is correctly able to communicate that datatype to Java. It is also noteworthy, that MAKEARRAY is defined such that in Object Rexx it is possible to use the DO...OVER construct to iterate over all elements of the Java array, irrespectible of its dimension, as well as the method SUPPLIER[23)] for iterating over all elements of the array.

The remainder of this section describes the public class “BSF” and its private subclasses “BSF_REFERENCE” and “BSF_ARRAY” as implemented and stored in the file “BSF.cls” as per May 2001.

2.2.5 Class “BSF”

This class defines the wrapper for Object Rexx proxy objects which represent Java obejcts registered in the BSF registry on the Java side. It allows for explicitly importing Java classes and sending Java methods transparently to the Java objects. Upon return of Java method invocations, the results are inspected and if a Java

[23)] The current implementation supplies the index part showing exactly the subscripts of the array element being supplied.

Method	Short Description
init **CLASS**	instantiates the default Rexx string for representing Java's "null" (i.e. ".NIL")
import **CLASS**	creates a subclass of BSF, places a reference to it into the local environment named according to the first argument and remembers the Java class it refers to
exit **CLASS**	calls BSF("exit", ...)
lookupBean **CLASS**	calls BSF("lookupBean", ...)
pollEventText **CLASS**	calls BSF("pollEventText", ...)
postEventText **CLASS**	calls BSF("postEventText", ...)
getStaticValue **CLASS**	calls BSF("getStaticValue", ...)
setRexxNullString **CLASS**	calls BSF("setRexxNullString", ...)
init	constructor, creates the Java object using BSF("registerBean", ...)
uninit	destructor, deregisters the Java object using BSF("unregisterBean", ...)
unknown	forwards all unknown messages to Java by invoking the Java methods using the message name unknown to the public class BSF. This causes BSF("invoke", ...) to be carried out.
bsf.addEventListener	calls BSF("addEventListener", ...) uses this object as argument
bsf.getFieldValue	calls BSF("getFieldValue", ...) uses this object as argument
bsf.setFieldValue	calls BSF("setFieldValue", ...) uses this object as argument
bsf.getPropertyValue	calls BSF("getPropertyValue", ...) uses this object as argument
bsf.setPropertyValue	calls BSF("setPropertyValue", ...) uses this object as argument
bsf.invoke	calls BSF("invoke", ...) uses this object as argument; note this method is intended to be used only, if a Java object happens to possess a method by the same name as implemented in the class BSF.

Table 2: Public Methods of the Public Class "BSF".

object is returned a new proxy of type "BSF_REFERENCE" will be dynamically created.

There are two ways to instantiate Java classes with the help of this Object Rexx Wrapper class:

Method	Short Description
dimension	returns the dimension of the array
items	returns the number of items in the array
at	retrieves the value stored at the given array position (index being 1-based!)
"[]"	same as method "at" above
put	stores the value at the given array position; **note:** the value *must* contain as the first word the datatype indicator for the value as indicated in table 1 above!
"[]="	same as method "put" above
makearray	returns a single dimensioned Object Rexx array; allows to enumerate all elements of the Java array object using the "DO...OVER" construct of Object Rexx
supplier	returns a supplier object rendering of the array

Table 3: Public Methods of the Class "BSF_ARRAY".

- First importing the Java class into Object Rexx and then sending the Object Rexx proxy class the Object Rexx message "NEW", or
- directly by instantiating the BSF-class, indicating the Java class and optionally the arguments needed for creating the Java objects.

In both cases an Object Rexx proxy object will get returned. The messages proxy objects receive, if unknown to the Object Rexx class "BSF" itself, will get repackaged and forwarded to the Java side by using the BSF-external function. This logic is implemented in the method "UNKNOWN".

All BSF-external functions are furthermore either implemented at the class level, if they do not pertain to individual Java objects but represent a generic function applicable to all instances or get the string "bsf." (including the dot!) prepended. Table 2 lists the methods defined in the class BSF and indicates which variant of the BSF external function is (procedurally) called.

2.2.6 Class “BSF_REFERENCE”

This is a specialization of the public class “BSF” and gets used to create proxy objects for Java objects, which got returned as a result of a Java method. This class is not public and is only meant to be used from within the file “BSF.cls”.

2.2.7 Class “BSF_ARRAY”

This is a specialization of the public class “BSF” and gets used to create proxy objects for Java objects, which got returned as a result of a Java method. This class is not public and is only meant to be used from within the file “BSF.cls”. It implements the semantics of Object Rexx arrays, *including* addressing the first element of an array with one (and not 0 as is the case with Java), being automatically translated to the Java indexing. Table 3 lists the methods implemented in this class.

3 SUMMARY AND OUTLOOK

This article introduced the reader to IBM's Bean Scripting Framework (BSF) which allows Java to call scripts in non-Java languages and to interact with such programs. As there was no Rexx or Object Rexx support available, a BSF-compliant Rexx engine was devised and developed, drawing heavily from previous work of the student Peter Kalender in the context of an Business Informatics seminar at the University of Essen. Due to the rewrite of major parts and addition of sophisticated functions this work can be regarded as mostly new compared to the student's work.

The reader should now be able to understand the BSF architecture and the available interfaces for Rexx and Object Rexx with the *bsf4rexx* [W3BSF4R] package in principle and be able to understand and follow the numerous test programs and sample programs with the aforementioned package.

This work allows to employ Rexx and Object Rexx in the context of the deployment of IBM's BSF, which not only allows its usage in creating Java Server Pages (JSP) with IBM's WebSphere product, but also to be used in the projects of the Apache organization [W3Apa] relating to the BSF technology, e.g. like XSLT or SOAP. Of course, it is now possible to use Rexx and Object Rexx to script any Java based application as well and to draw from the rich set of ideas and knowledge of Rexx coders who have been developing scripts for different environments in the past 20 years.

From the perspective of a Rexx or Object Rexx programmer the support for IBM's BSF allows them to immediately regard the Java runtime environment (JRE) as a huge, ported set of external functions available on any platform Java and Rexx runs on. This in turn means that this combination of Rexx and Java effectively runs on all platforms! Only time will tell, how fast Rexx programmers will adopt this innovative technology for new creative problem solutions and multi-platform deployment of such Rexx programs.

Further discussions are delegated to and expected to take place in the Internet newsgroup <news:comp.lang.rexx>.

4 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.

[Kal01] Kalender P.: “A Concept for and an Implementation of the Bean Scripting Framework for Rexx”, Seminar paper, University of Essen, MIS and Software Engineering Department, February 2001. URL (2001-04-22): `http://nestroy.wi-inf.uni-essen.de/Lv/seminare/ws0001/PKalender/Seminararbeit.pdf`

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

[W3Alpha] Homepage of IBM’s alphaworks projects, URL (2001-04-22): `http://www.alphaworks.ibm.com/`

[W3Apa] Homepage of the open source Apache organization, URL (2001-05-05): `http://www.apache.org/`

[W3BSF] Homepage of IBM’s “Bean Scripting Framework” (BSF), URL (2001-04-22): `http://oss.software.ibm.com/developerworks/projects/bsf`

[W3BSF4R] Homepage of the “bsf4rexx” package, URL(2001-04-22): `http://nestroy.wi-inf.uni-essen.de/Forschung/rgf/Entwicklung.html`

[W3Java] Java homepage, URL (2001-04-22): `http://http://www.sun.com/java`

[W3NetRexx] NetRexx homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2001-04-22): `http://www2.hursley.ibm.com/netrexx/`

[W3ObjRexx] Object Rexx homepage of IBM, URL (2001-04-22): `http://www.ibm.com/software/ad/obj-rexx/`

[W3Rexx] Rexx homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2001-04-22): http://www2.hursley.ibm.com/rexx/

[W3RexxLA] Rexx homepage of the “Rexx Language Association”, URL (2001-04-22): http://www.RexxLA.org

[W3Rhino] Rhino homepage, URL (2001-04-22): http://www.mozilla.org/rhino

Date of Article: 2001-05-06.

Published in: Proceedings of the „12th International Rexx Symposium“, Triangle Research Park, North Carolina, USA, April 30th - May 2nd, The Rexx Language Association, Raleigh N.C. 2000.

Presented at: „12th International Rexx Symposium“, Raleigh, North Carolina, USA, April 30th - May 2nd, 2001 24-26.