"The 2010 Edition of BSF4ooRexx"

2010 International Rexx Symposium
Amsterdam/Almere, Netherlands (December 2010)

© 2010 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (http://www.wu.ac.at)
BSF4ooRexx

• External Rexx function package
  – Allows to interact with the Java runtime environment (JRE)
    • Exploit functionality of Java classes
    • Exploit functionality of Java objects
  – ooRexx 4.0.1 and later
  – Package "BSF.CLS"
    • Camouflages Java as ooRexx
    • Supplies class BSF and public routines
  – "Everything that is available in Java becomes directly available to ooRexx!"
BSF4ooRexx
An Example

dim=.bsf~new("java.awt.Dimension", 100, 200)
say dim~toString

::requires BSF.CLS -- get Java support

Output:
java.awt.Dimension[width=100,height=200]
Things to Know About Java, 1

• Strictly typed language
  – Primitive types
    • boolean, byte, char, short, int, long, float, double
  – Object-oriented types
    • Any Java class, e.g.
      – java.awt.Dimension, java.lang.String, java.lang.System, ...
    • Wrapper classes for primitive types
      – java.lang.Boolean, java.lang.Byte, java.lang.Character,
        java.lang.Short, java.lang.Integer, java.lang.Long,
        java.lang.Float, java.lang.Double
      – "Boxing": wraps up a primitive value into a wrapper object
      – "Unboxing": retrieves a primitive value from its wrapper object
• Case sensitive
  – Upper- and lowercase significant!

• Classes organized in packages
  – Package names may be compound
    • E.g. "java.lang"
  – Fully "qualified class name" includes package name
    • e.g. "java.lang.String"
  – "Unqualified class name"
    • e.g. "String"
A Java class may consist of
- Fields (comparable to ooRexx attributes) and
- Methods (comparable to ooRexx methods)

Fields and methods
- Static fields and static methods
  - Sometimes dubbed "class fields" and "class methods"
  - Available to the class object and its instances
- Otherwise "instance methods"
  - Only available to instances of a Java class
Things to Know About Java, 4

- A Java class, its fields and methods may be:
  - "public"
    - These can be accessed by the "world" (everyone)
  - "private"
    - Only accessible within the Java class
  - "protected"
    - Only accessible within Java classes of the same package and subclasses
  - None of the above modifier given
    - Only accessible within Java classes of the same package, but to no one else
Things to Know About Java, 5

- Excellent documentation ("Javadoc")
  - Extensive set of interlinked HTML documents
  - Created right from the comments in Java sources
  - Can be studied on the Internet

- **First** start out with the older version Java 1.4
  - Overview:
    - [http://download.oracle.com/javase/1.4.2/docs/index.html](http://download.oracle.com/javase/1.4.2/docs/index.html)
  - All Java classes:
    - [http://download.oracle.com/javase/1.4.2/docs/api/](http://download.oracle.com/javase/1.4.2/docs/api/)

- **Then** study the latest version, e.g. Java 1.6 ("Java 6")
  - [http://download.oracle.com/javase/6/docs/index.html](http://download.oracle.com/javase/6/docs/index.html)
  - [http://download.oracle.com/javase/6/docs/api/](http://download.oracle.com/javase/6/docs/api/)
A Javadoc Example

Class XyzType

```java
java.lang.Object
   + XyzType
```

public class XyzType
extends java.lang.Object

### Field Summary

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
</tr>
<tr>
<td>counter</td>
</tr>
</tbody>
</table>

### Constructor Summary

- `XyzType()`
- `XyzType(java.lang.String initialValue)`

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td><code>getInfo()</code></td>
</tr>
<tr>
<td>void</td>
<td><code>setInfo(java.lang.String sValue)</code></td>
</tr>
</tbody>
</table>
BSF.CLS

Camouflages Java as ooRexx

• ooRexx class "BSF"
  – Allows to create Java objects
  – Needs at least fully qualified Java class name

• Invoking Java methods
  – Just send the name of the method to the Java object
    • Supply the arguments as documented, if any
      – Type conversions between ooRexx and Java are done automatically by BSF4ooRexx, if necessary
      – Return values are automatically converted by BSF4ooRexx, if necessary
Example Using Java Class "XyzType", 1

```rexx
BSF: new("XyzType")
say "o~getInfo:" o~getInfo
o~setInfo("Hello, from ooRexx...")
say "o~getInfo:" o~getInfo
::requires BSF.CLS -- get Java support
```

**Output:**

```
o~getInfo: The NIL object
o~getInfo: Hello, from ooRexx...
```
BSF.CLS

Camouflages Java as ooRexx

• ooRexx class "BSF" (continued)
  – Allows to create Java objects
  – Needs at least fully qualified Java class name

• Possible arguments for creating Java objects
  – Can be found by studying the "Constructor" section in the Javadocs
  – Supply the arguments as documented after the fully qualified Java class name argument
    – Type conversions between ooRexxx and Java are done automatically by BSF4ooRexxx, if necessary
Example Using Java Class "XyzType", 2

```rexx
do BSF new("XyzType", "This value was supplied at Java object creation.")
say "o~getInfo:" o~getInfo

setInfo("Hello, from ooRexx...")
say "o~getInfo:" o~getInfo

::requires BSF.CLS -- get Java support
```

**Output:**

```
o~getInfo: This value was supplied at Java object creation.
o~getInfo: Hello, from ooRexx...
```
BSF.CLS
Camouflages Java as ooRexx

• Allows to import any Java class
  – `bsf.import(JavaClassName)`
    • Java class name
      – Use of the exact case is mandatory!
      – Java class name must be fully qualified!

  – Imported class can be treated as if it was an ooRexx class
    • Allows to use the ooRexx "new"-method to create instances of the imported Java class
      – Possible arguments for creating Java objects can be found by studying the "Constructor" section in the Javadoc.
Example Using Java Class "XyzType", 3

```rexx
clz=BSF.import("XyzType")
o=clz~new("This value was supplied at Java object creation.")
say "o~getInfo:" o~getInfo

o~setInfo("Hello, from ooRexx...")
say "o~getInfo:" o~getInfo

::requires BSF.CLS -- get Java support
```

Output:

```
o~getInfo: This value was supplied at Java object creation.
o~getInfo: Hello, from ooRexx...
```
BSF.CLS
Camouflages Java as ooRexx

• Accessing, setting Java fields
  – ooRexx treats public fields as ooRexx attributes
  – Java "get" and "set" pattern methods for Java fields honored by BSF4ooRexx
    • Just use the field name following "get" and "set" only
  – Static fields can be accessed via the
    • Java class object or
    • any of its instances
Example Using Java Class "XyzType", 4

```rexx
clz=BSF.import("XyzType")
say "clz~counter:" clz~counter

o=clz~new("This value was supplied at Java object creation.")
say "clz~counter:" clz~counter
say "o ~counter:" o ~counter

say "o~getInfo:" o~getInfo

o~setInfo("Hello, from ooRexx...")
say "o~getInfo:" o~getInfo

clz~~new~~new~~new
say "clz~counter:" clz~counter "/" "o~counter:" o ~counter

::requires BSF.CLS -- get Java support
```

Output:

```
clz~counter: 0
clz~counter: 1
o ~counter: 1
o~getInfo: This value was supplied at Java object creation.
o~getInfo: Hello, from ooRexx...
oclz~counter: 4 / o~counter: 4
```
Example Using Java Class "XyzType", 5

```rexx
clz=BSF.import("XyzType")
say "clz~counter:" clz~counter

o=clz~new("This value was supplied at Java object creation.")
say "clz~counter:" clz~counter
say "o  ~counter:" o  ~counter

say "o~getInfo:" o~getInfo

o~info="Hello, from ooRexx..."
say "o~info:" o~info

clz~~new~~new~~new
say "clz~counter:" clz~counter "/" "o~counter:" o  ~counter

::requires BSF.CLS      -- get Java support
```

Output:

clz~counter: 0
clz~counter: 1
o  ~counter: 1
o~getInfo: This value was supplied at Java object creation.
o~info: Hello, from ooRexx...
o~info: Hello, from ooRexx...
clz~counter: 4  /  o~counter: 4
• About respecting case
  – Case of fully qualified Java class name
    • Always significant!
  – Case of fields and method names *insignificant*
    • Eases coding enormously
Example Using Java Class "XyzType", 6

```
clz=BSF.import("XyzType")
say "clz~COUNTER:" clz~COUNTER

o=clz~new("This value was supplied at Java object creation.")
say "clz~Counter:" clz~Counter
say "o ~counter:" o ~counter

say "o~getinfo:" o~getinfo

o~info="Hello, from ooRexx..."
say "o~info:" o~info

clz~~new~~new~~new
say "clz~Counter:" clz~Counter "/" "o~Counter:" o ~Counter

::requires BSF.CLS -- get Java support
```

Output:

```
clz~COUNTER: 0
clz~Counter: 1
o ~Counter: 1
o~getinfo: This value was supplied at Java object creation.
o~info: Hello, from ooRexx...
o~info: Hello, from ooRexx...
clz~Counter: 4 / o~Counter: 4
```
Creating Java Arrays

- Java arrays
  - Strictly typed
  - Fixed capacity

- Public routine "bsf.createJavaArray(...)"
  - First argument gives the Java type
    - Fully qualified Java class name or
    - Java class object
  - Each further argument is an integer value, denoting the maximum elements in that dimension
Creating Java Arrays, 1

- Java arrays
  - Strictly typed
  - Fixed capacity
  - Indices start with value "0"

- Public routine "bsf.createJavaArray(...)
  - Arguments
    - First argument gives the Java type
      - Fully qualified Java class name or Java class object
    - Each further argument is an integer value, denoting the maximum elements in that dimension
Creating Java Arrays, 2

- Public routine "bsf.createJavaArray(...)"
  - Resulting Java array can be used as if it was an ooRexx array object!
    - Indices start at "1" as with ooRexx arrays!
    - Possesses the fundamental ooRexx array methods like "AT", "[]", "PUT", "[]=", "supplier", and "makeArray"
    - Can be used in ooRexx "DO ... OVER" loops
Example of Creating a Java Array

```plaintext
-- create a two-dimensional (5x10) Java Array of type String
arr=.bsf~bsf.createJavaArray("java.lang.String", 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

::requires BSF.CLS -- loads Java support
```

Output:

First Element in Java array.
Last Element in Java array.
BSF4ooRexx

BSFCreateRexxProxy, 1

• RexxProxy
  – A Java object that proxies an ooRexx object
  – Any method invocations on the Java object will be forwarded as an ooRexx message to the proxied ooRexx object
    • All arguments supplied to the Java method are forwarded in the same sequence with the ooRexx message
    • BSF4ooRexx may append an additional argument, "slotDir" (an ooRexx directory object) to the ooRexx message, which will contain information about the Java method invocation
• RexxProxy
  - BSFCreateRexxProxy(rexxObj [, userData])
    • Creates and returns a Java object that proxies "rexxObj"
    • If "userData" (any Rexx object) supplied, then it will be added to the "slotDir" directory
  - BSFCreateRexxProxy(rexxObj [, [userData], jiClz[, ...]])
    • "jiClz" can be one or more Java interface classes the returned RexxProxy can be used for!
  - BSFCreateRexxProxy(rexxObj [, [userData], jaClz[, arg[,...]]])
    • "jaClz" is an abstract Java class, "arg" can be one or more arguments for creating an instance of it
Example RexxProxy, 1

```rexx
rexxObj=.myClass~new
rexxObj~hello
say "---"
rp=BSFCreateRexxProxy(rexxObj) -- create a Java RexxProxy object
rp~sendMessage0("hello") -- send via Java
::requires BSF.CLS -- get Java support
::class myClass
::method hello
   say "hello from" pp(self)
```

Output:

```
hello from [a MYCLASS]
---
hello from [a MYCLASS]
```
Example RexxProxy, 2

```ruby
rexxObj=.myClass~new
rexxObj~hello
say "---"

userData="This is some Rexx string."  -- sent only if invoked via Java
rp=BSFCreateRexxProxy(rexxObj,userData)  -- create a Java RexxProxy object
rp~sendMessage0("hello")  -- send via Java

::requires BSF.CLS  -- get Java support

::class myClass
::method hello
  use arg slotDir  -- available only, if called from Java
  if slotDir~isA(.directory) then
    say "hello from" pp(self) "userData:" pp(slotDir~userData)
  else
    say "hello from" pp(self)
```

Output:

```
hello from [a MYCLASS]
---
hello from [a MYCLASS] userData: [This is some Rexx string.]
```
BSF4ooRexx
Roundup, 1

- External Rexx function package
  - Takes advantage of ooRexx 4.0.1 and later
  - Allows Interacting with Java classes and objects

- "BSF.CLS"
  - Camouflages Java as ooRexx
  - Allows easy creation of Java objects
    - Java class name must be fully qualified and in exact case
  - Allows sending ooRexx messages to Java objects
    - No strict casing, no strict typing
BSF4ooRexx
Roundup, 2

• BSFCreateRexxProxy()
  – Wraps up an ooRexx object in a Java object
  – Allows to send messages to ooRexx from Java
  – Very powerful if used with Java interface classes or Java abstract classes
    • Java abstract methods can be implemented in ooRexx!
Addendum

• Please note
  – The following slides explain a built-in mechanism to BSF4ooRexx that you will probably never need to use
  – However, should you ever run into a situation where case or type becomes important for BSF4ooRexx to work, then the following slides will help you solve such a challenge easily
Addendum

Extremely Rare Cases, 1

• Possible (extremely!) rare case problem
  – Possible that a Java class has different fields and methods with the same name, but with different cases
    • For Java these are different fields and methods
    • BSF4ooRexxx does not distinguish by default

• Possible (extremely!) rare type problem
  – Possible that a Java class has different methods with the same name and type-convertible primitive arguments, but with different behaviour
To solve such rare problems

- Wrap up primitive types using the public routine
  - box("typeIndicator",value)
- "Type indicators" are Rexx strings
  - Indicate primitive types must be used
    - "BOolean", "BYte", "Character", "SHort", "Integer", "Long", "Float", "Double"
- Special type indicators
  - "STring", turn into a Java string
  - "Object", value is a non-primitive value (only used for methods, see next slide)
Addendum

Extremely Rare Cases, 3

- To solve such rare problems the following methods are available for Java objects
  - Field related
    - `bsf.getFieldValueStrict(exactName)`
    - `bsf.setFieldValueStrict(exactName, [typeIndicator,] newValue)`
  - Method related
    - `bsf.invokeStrict(exactMethodName [, typeIndicator, argument]...)`
      - "typeIndicator" precedes each argument
  - Constructor related
    - If Java class was imported using `bsf.import(...)`, then
      - in addition to "new" the method "newStrict" is available, which expects each argument to be preceded by a "typeIndicator"
Example of Using "strict" BSF-methods

```plaintext
clz=BSF.import("XyzType")
say "clz-counter (strict):" clz-bsf.getFieldValueStrict("counter")

o=clz~newStrict("String", "This value was supplied at Java object creation.")
say "clz-counter (strict):" clz-bsf.getFieldValueStrict("counter")
say "o ~counter (strict):" o ~bsf.getFieldValueStrict("counter")

say "o~getInfo (strict):" o~bsf.invokeStrict("getInfo")

o~bsf.invokeStrict("setInfo", "String", "Hello, from ooRexx...")
say "o~getInfo (strict):" o~bsf.invokeStrict("getInfo")

clz~~newStrict~~new~~newStrict
say "clz-counter (strict):" clz-bsf.getFieldValueStrict("counter")
say "o~counter (strict):" o ~bsf.getFieldValueStrict("counter")

::requires BSF.CLS -- get Java support
```

Output:

```
clz-counter (strict): 0
clz-counter (strict): 1
o ~counter (strict): 1
o~getInfo (strict): This value was supplied at Java object creation.
o~getInfo (strict): Hello, from ooRexx...
clz-counter (strict): 4
o~counter (strict): 4
```