BSF4ooRexx850

The Bean Scripting Framework for ooRexx

Business Programming 2

Windows GUIs (AWT)	Sockets SSL/TLS	XML SAX/DOM JSON	Scripting AOO/LO (UNO)	Rexx Script Engine	Portable GUIs (JavaFX)	Java Web Server (Tomcat)	Java Classes written in Rexx style

Prof. Rony G. Flatscher

Java

- Programming language with the following notable features
  - Compiles to machine instructions (*"bytecode"*) of an *artificial processor*
  - Needs a "Java virtual machine (JVM)" to execute the bytecodes
    - JVMs are available for all important operating systems and hardware architectures
    - *Hence, a Java class or a Java program, once compiled can be run everywhere!*
  - Distributed with a (huge) "Java runtime environment (JRE)"
    - A *huge Java class library* that offers everything that an application may possibly need
      - E.g. Socket classes for Internet programming, GUI classes for graphical user interfaces, ...
    - Uncountable third party Java class libraries, most available as open-source (e.g. ASF)
  - Most important programs get programmed with Java (even Android applications!)
  - Many professional applications that are not programmed in Java offer Java APIs
    - E.g. SAP, OpenOffice/LibreOffice, ...
- Hence Java is truly a programmer's "treasure trove" for all operating systems!

BSF4ooRexx850

- External Rexx function package
  - Allows to interact with the Java runtime environment (JRE)
    - Exploit functionality of Java classes
    - Exploit functionality of Java objects
  - ooRexx 5.0 and later
  - Package "BSF.CLS"
    - Camouflages Java as ooRexx (Java appears to be dynamic and message based)
    - Supplies class BSF and public routines
- "Everything that is available in Java becomes directly available to ooRexx !"
  - Java: "write once, run everywhere!"
    - Windows, MacOS, Linux, ...

BSF4ooRexx850: 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]
```

Downloading Java (Usually Free and Open-source)

- JRE versus JDK
  - JRE: "**J**ava **R**untime **E**nvironment", no compiler
  - JDK: "**J**ava **D**evelopment **K**it", compiler & tools
- Java/JDK 8 LTS ("long term support")
  - Released spring 2014, supported until 2030 (Oracle, Azul), 2031 (BellSoft)
- Java/JDK 25 LTS ("long term support", "modular Java")
  - Release September 2025, supported at least until 2033 (Oracle, Azul, BellSoft)
- Suggestion: download OpenJDK *with JavaFX* support, e.g.
  - Scroll down to see all versions pick the *JavaFX* installation package
    - **Full JDK:** <https://bell-sw.com/pages/downloads/> ("Liberica", 2025-08-18)
    - **JDK FX:** <https://www.azul.com/downloads/> (2025-08-18)

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

Things to Know About Java, 2

- 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"

Things to Know About Java, 3

- 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 modifiers given ("package private")
    - 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, search e.g. with

```
javadoc 8 java.awt.Dimension
javadoc 8 Dimension
javadoc 25 java.awt.Dimension
javadoc 25 Dimension
```

- Documentation can be downloaded to local computer, e.g.
  - Java/JDK 8 LTS (LTS: acronym for "long term support"), last non-module version:
    - <https://www.oracle.com/java/technologies/javase-jdk8-doc-downloads.html> (2025-12-04)
  - Java/JDK 25 LTS (LTS: acronym for "long term support"):
    - <https://www.oracle.com/java/technologies/javase-jdk25-doc-downloads.html> (2025-12-04)

A Javadoc Example (JDK8LTS)

Class XyzType

java.lang.Object
    XyzType

```
public class XyzType
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
static int	counter

Constructor Summary

Constructors

Constructor and Description
XyzType()
XyzType(java.lang.String initialValue)

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
java.lang.String	getInfo()
void	setInfo(java.lang.String aValue)

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 BSF4ooRexx850, if necessary
    - Return values are automatically converted by BSF4ooRexx850, if necessary

BSF4ooRexx850: Java Class "XyzType", 1

```
o=.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 without “XyzType.class” available:

```
Error 40.900:  BSF4ooRexx/routine/BSF(), error 3: Java exception occurred:
[org.apache.bsf.BSFException: BSF4ooRexx subfunction "new": while attempting to load class
'XyzType', threw: [org.apache.bsf.BSFException: BSF4ooRexx subfunction "new": while attempting to
load class 'XyzType', threw exception: [java.lang.ClassNotFoundException: XyzType]]].]
```

Output with “XyzType.class” available:

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

XyzType.java

- "XyzType.java" (Source Code)

```
/*  This is the most important class of all!
 *  Even if one does not believe this, this is so! ;)
 *  &Ouml;sterreich, Wien.
 */

public class XyzType    // example class for demonstrating BSF4Rexx
{
        // constructors of this class (same name as class!)
    public XyzType () {       // constructor without arguments
        counter=counter+1;    // increase counter
    }

    public XyzType (String initialValue) {   // constructor with argument
        this();               // invoke constructor above (no argument)
        info=initialValue;    // save initial value
    }
        // keyword "static": class fields (attributes) and class methods
    static public int counter=0;    // field: will count # of instances


        // instance fields (attributes) and instance methods
    private String info = null; // field: no initial value per default

    public String getInfo () {  // accessor (getter) method (function)
        return info;            // return whatever "info" points to
    }

    public void setInfo (String aValue) {   // setter method (function)
        info=aValue;            // save received value with "info"
    }
}
```

```
Compile Java source file with the Java compiler
"javac" into "byte code":

javac XyzType.java

creates:  XyzType.class
```

```
Create Java documentation with the utility
"javadoc", place resulting files into the
directory named docs:

javadoc -d docs XyzType.java

load documentation from: docs\index.html
```

BSF.CLS: Camouflages Java as ooRexx

- ooRexx class "**BSF**"
  - 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 ooRexx and Java are done automatically by BSF4ooRexx, if necessary

BSF4ooRexx850: Java Class "XyzType", 2

```
o=.BSF~new("XyzType", "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

- 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 Java class can be treated as if it were 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 Javadocs

BSF4ooRexx850: Java Class "XyzType", 3

```
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

BSF4ooRexx850: Java Class "XyzType", 4

```
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...
clz~counter: 4 / o~counter: 4
```

BSF4ooRexx850: Java Class "XyzType", 5

```
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...
clz~counter: 4 / o~counter: 4
```

BSF.CLS

- About respecting case
  - Case of fully qualified Java class name
    - Always significant!
- Case of fields and method names insignificant!
  - Eases coding considerably

Prof. Rony G. Flatscher

BSF4ooRexx850: 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...
clz~CounteR: 4 / o~cOUNTEr: 4
```

BSF.CLS: 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

BSF.CLS: 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 therefore used in ooRexx "**DO ... OVER**" and "**DO WITH ... OVER**" loops

BSF.CLS: Creating a Java Array

```
-- 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 o over arr      -- loop over elements in array (makearray)
   say o
end
say

do with index i item o over arr  -- loop over elements in array (supplier)
   say i":" o
end

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

Output:

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

1,1: First Element in Java array.
5,10: Last Element in Java array.
```

BSF4ooRexx850: BSFCreateRexxProxy, 1

- RexxProxy
  - A *Java object* that proxies an ooRexx object
  - Allows Java to send messages to ooRexx objects
  - 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
    - BSF4ooRexx850 always appends an additional argument, "**slotDir**" (an ooRexx directory object) to the ooRexx message, which will contain information about the Java method invocation

BSF4ooRexx850: BSFCreateRexxProxy, 2

- 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

BSF4ooRexx850: RexxProxy, 1

```
rexxObj=.myClass~new
rexxObj~hello
say "---"
jrp=BSFCreateRexxProxy(rexxObj)    -- create a Java RexxProxy object
jrp~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]
```

BSF4ooRexx850: RexxProxy, 2

```
rexxObj=.myClass~new
rexxObj~hello
say "---"
userData="This is some Rexx string."      -- sent only if invoked via Java
jrp=BSFCreateRexxProxy(rexxObj,userData)  -- create a Java RexxProxy object
jrp~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.]
```

BSF4ooRexx850: Roundup, 1/2

- External Rexx function package
  - BSF4ooRexx version *850* needs at least Java *8* or later, and ooRexx *5.0* or later
  - Allows interacting with Java classes and objects
- "**BSF.CLS**"
  - An ooRexx package containing public classes and public routines for ooRexx
  - Its public class "BSF" 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 necessary!

BSF4ooRexx850: Roundup, 2/2

- 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 BSF4ooRexx850 to work, then the following slides will help you solve such a challenge easily

Addendum: Extremely Rare Cases, 1

- Possible problem (although very unlikely to materialize)
  - 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, because the case of their names is different
    - By default BSF4ooRexx850 does not distinguish the case of field and method names
- Possible (extremely rare, almost non-existent) type problem
  - Possible that a Java class has different methods with the same name and type-convertible primitive arguments, but with different behaviour
  - Solution: use the public routine **box("typeIndicator",value)** from the **BSF.CLS** package
    - Easy to create and therefore force a specific Java type for primitive values

Addendum: Extremely Rare Cases, 2

- 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
    - "**BO**olean", "**BY**te", "**C**haracter", "**SH**ort", "**I**nteger", "**L**ong", "**F**loat", "**D**ouble"
  - Special type indicators
    - "**ST**ring", turn into a Java string
    - "**O**bject", 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**" preceeds 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**"

Addendum: Using "strict" BSF-methods

```
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
```