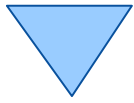




"The Vienna Version of BSF4Rexx"

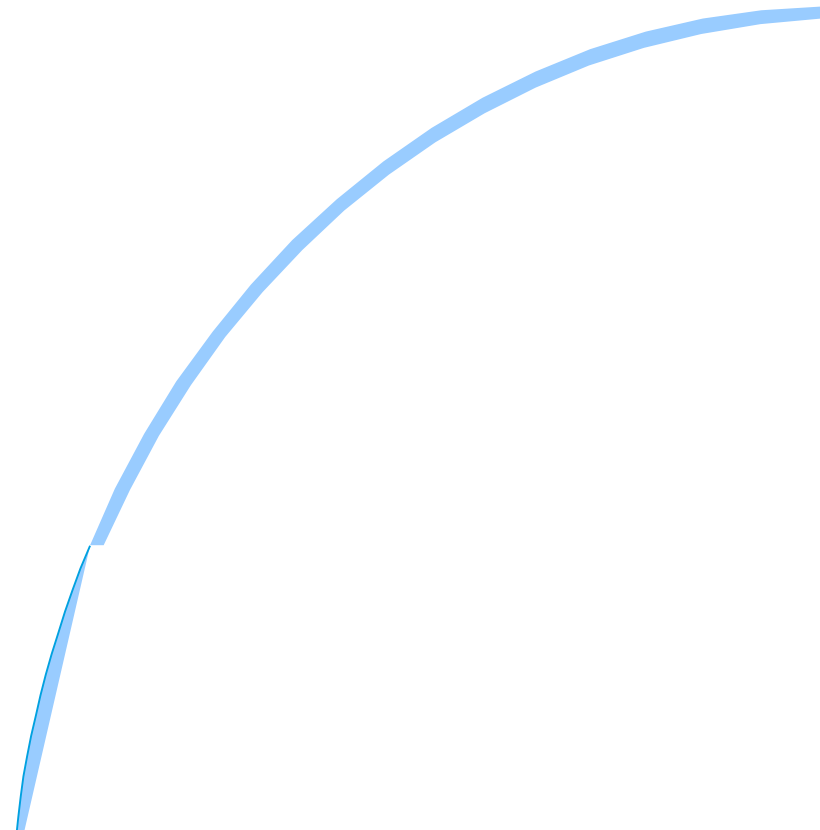
2006 International Rexx Symposium
Austin, Texas, U.S.A. (April 2006)

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



Agenda (Vienna Version, 2006)

- Brief History
- Architecture
- Changes
 - Examples
- New features
 - Examples
- Roundup and Outlook

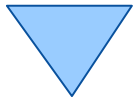


▼ BSF4Rexx History, 1

- Wintersemester 2000/01
 - Seminar assignment at the University of Essen
 - Proof of concept by a student (Peter Kalender)
- Spring 2001
 - Introduction of a re-write and w.r.t. BSF complete version of "BSF4Rexx" to the RexxLA
 - Ongoing work and improvements


▼ BSF4Rexx History, 2

- Spring 2003
 - Introduction of the "Augsburg" version of BSF4Rexx to the RexxLA
 - Bug fixing
 - Added a few external Rexx functions to the external function package "BSF4Rexx.dll"
 - E.g. allows to demand load Java on Linux and Windows



Agenda (from 2003)

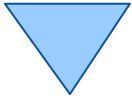
- Revealing the *real* Title
- Brief Architecture
 - The "Essener" Version of BSF4Rexx (2001)
 - The "Augsburg" Version of BSF4Rexx (2003)
- An example
 - A Java program
 - A Rexx program
- Additional new features
- Roundup



The Largest External Function Package for Rexx on Earth !

[And already ported to
all important
operating systems and hardware platforms!]

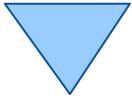




BSF

<http://jakarta.apache.org/bsf>

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

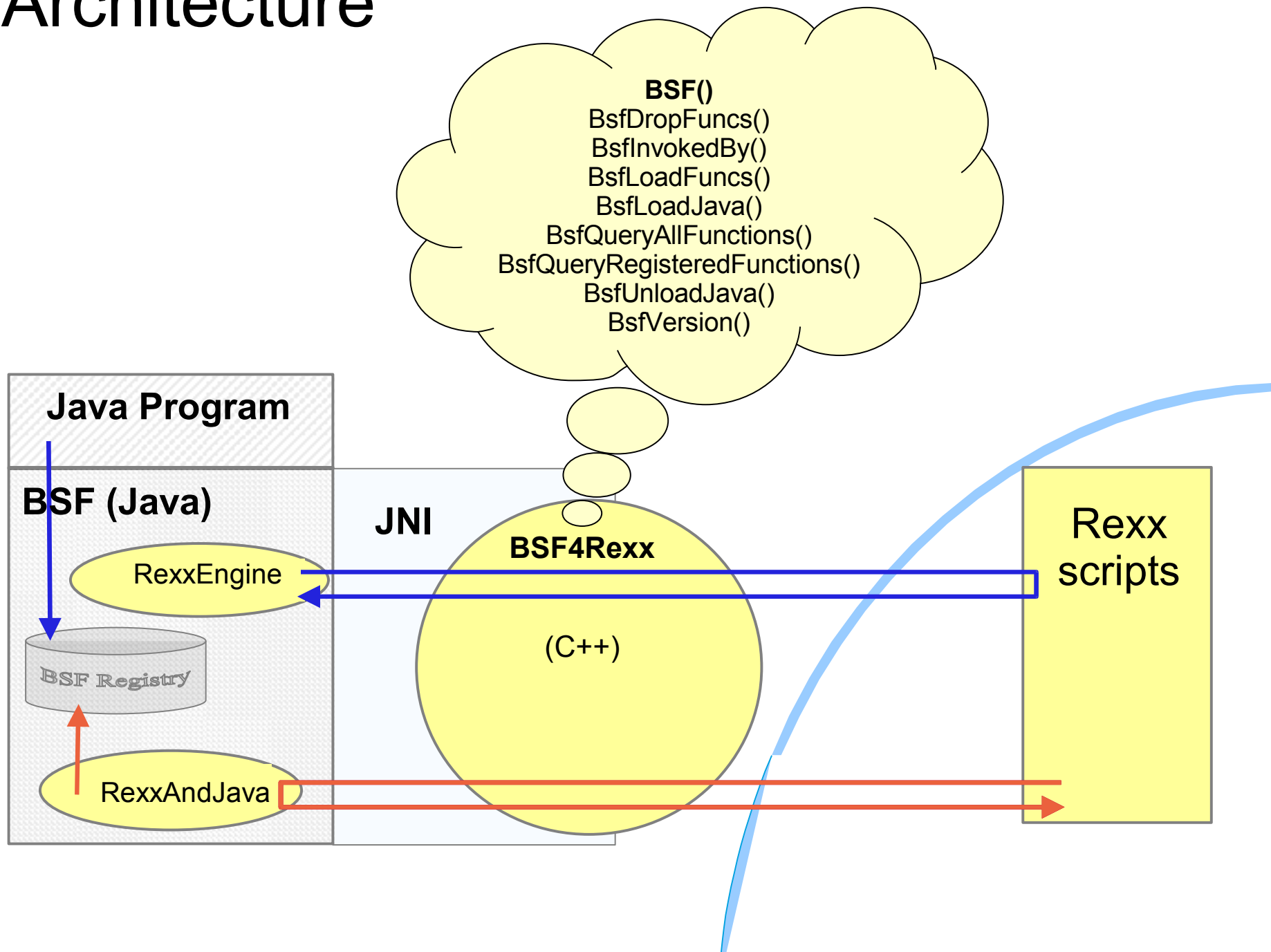



BSF4Rexx

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

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

BSF4Rexx Architecture





Java Invoking a Rexx Script An Example

```
import com.ibm.bsf.*;    // BSF support
import java.io.*;       // exception handling

public class TestSimpleExec {

    public static void main (String[] args) throws IOException
    {
        try {
            BSFManager mgr = new BSFManager ();
            BSFEngine rexx = mgr.loadScriptingEngine("rexx");
            String rexxCode = "SAY 'Rexx was here!'";

            rexx.exec ("rexx", 0, 0, rexxCode);

        } catch (BSFException e) { e.printStackTrace(); }
    }
}
```

Output:

Rexx was here!



BSF4Rexx

A Rexx Script Interfacing with Java

```
/* "getJavaVersion.rex": 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')
```

Invoking the program either with:

```
rexex getJavaVersion.rex
```

or:

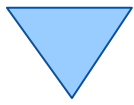
```
java org.rexxla.bsf.RexxDispatcher getJavaVersion.rex
```

or (shorthand of the above):

```
{rexexj.cmd|rexexj.sh} getJavaVersion.rex
```

Possible Output:

```
java.version: 1.5.0_06
```

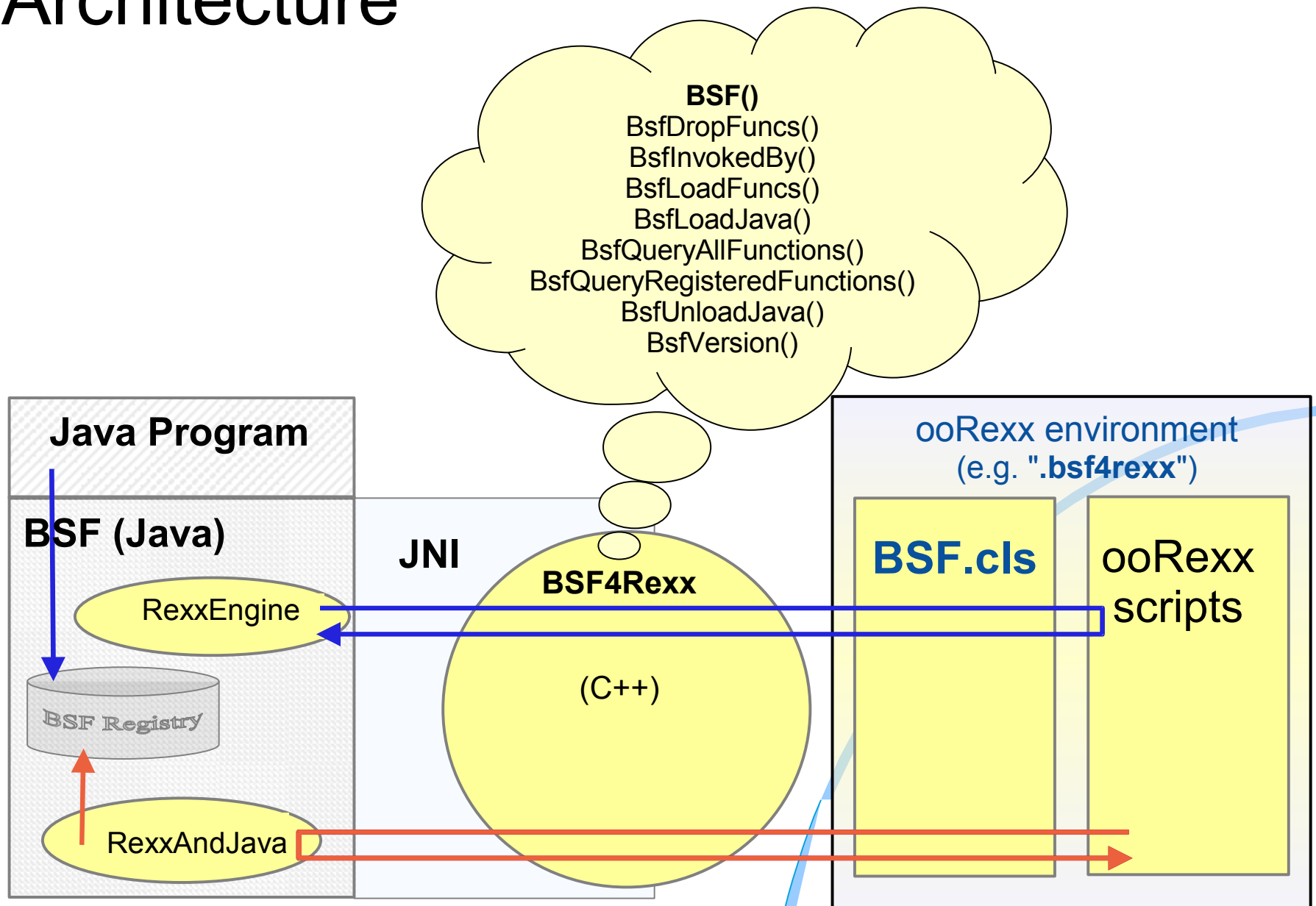


BSF.CLS

Entering ooRexx

- **BSF.CLS**
 - An ooRexx module containing
 - Supporting BSF via the proxy class **BSF**
 - Supporting BSF routines, e.g. `bsf.import(...)`
 - Services like making the most important and pre-registered Java classes directly available via the environment symbol **.bsf4rexx**
 - Will load Java transparently, if not yet loaded
 - Rexx programs

BSF4Rexx with **BSF.CLS** Architecture



BSF4Rexx with BSF.CLS

A Rexx Script Interfacing with Java, 1

```
/* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */  
  
call bsf.cls          /* load the Java support */  
say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')
```

```
/* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */  
  
say "java.version:" bsf('invoke', 'System.class', 'getProperty', 'java.version')  
::requires bsf.cls /* load the Java support */
```

Invoking the program either with:

```
rexx getJavaVersion.rex
```

or:

```
java org.rexxla.bsf.RexxDispatcher getJavaVersion.rex
```

or (shorthand of the above):

```
{rexxj.cmd|rexxj.sh} getJavaVersion.rex
```

Possible Output:

```
java.version: 1.5.0_06
```

BSF4Rexx with BSF.CLS

A Rexx Script Interfacing with Java, 2

```
/* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */  
  
s=bsf.import('java.lang.System') /* import the Java class 'java.lang.System' */  
say "java.version:" s~getProperty('java.version')  
  
::requires bsf.cls /* load the Java support */
```

```
/* "getJavaVersion.rex": classic Rexx version, querying the installed Java version */  
  
say "java.version:" .bsf4rex~system.class ~getProperty('java.version')  
  
::requires bsf.cls /* load the Java support */
```

Invoking the program either with:

```
rexex getJavaVersion.rex
```

or:

```
java org.rexxla.bsf.RexxDispatcher getJavaVersion.rex
```

or (shorthand of the above):


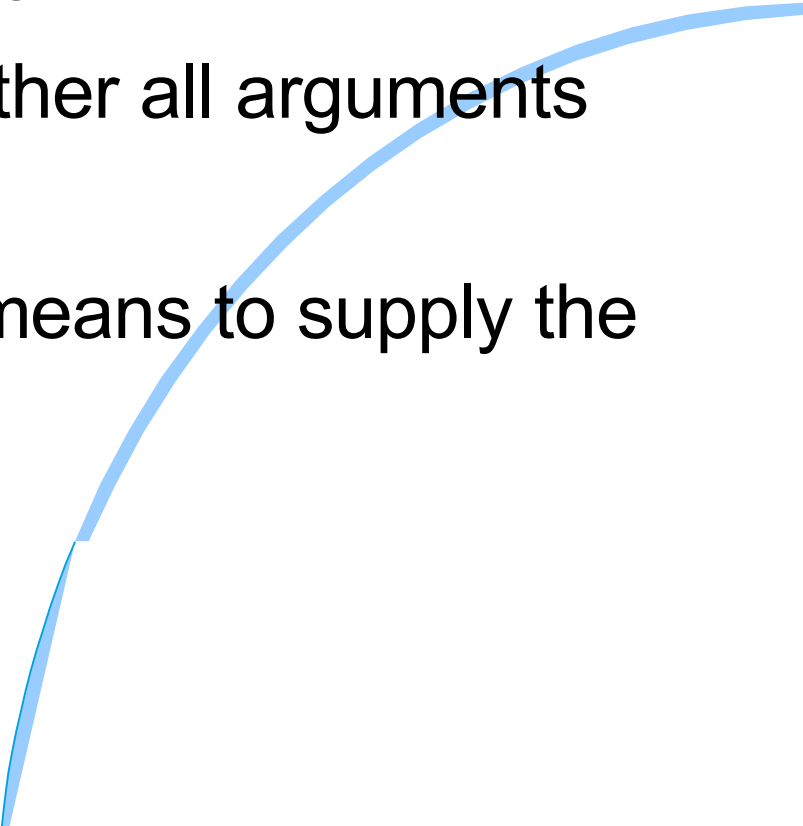
```
{rexexj.cmd|rexexj.sh} getJavaVersion.rex
```

Possible Output:

```
java.version: 1.5.0_06
```




Java's Strong Typing

- 
- Every variable needs to be typed
 - Java compiler must have access to type
 - Java compiler checks whether all variables are used according to their type
 - Java compiler checks whether all arguments are of the correct type
 - Hence interfacing with Java means to supply the correct types!
- 



BSF4Rexx Type Indicators, 1

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



BSF4Rexx Type Indicators, 2

New Feature

- Starting with the Vienna version of BSF4Rexx no need to indicate Java types anymore
 - Makes it simpler to use Java
 - BSF4Rexx will figure out the correct types and supply Java with them!
 - Still, strongly typed subfunctions are made available and start with the word "Strict"
 - May be needed in very rare circumstances

BSF4Rexx Type Indicators, 3

BSF.CLS – New Feature

- Sometimes one needs to supply primitive datatypes embedded in Java classes
- Public routines `box()`, `unbox()`

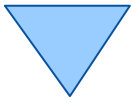
```
javaObject=box(TypeIndicator, primitiveValue)
primitiveValue=unbox(javaObject)
```

```
javaObject=box('Long', '123456789012') /* wrap a long value in a Java object */
say javaObject /* name of object in BSF registry */
say javaObject~toString /* string representation by Java class */
primitiveValue=unbox(javaObject) /* Rexx string */
say primitiveValue

::requires bsf.cls /* load the Java support */
```

Possible Output:

```
java.lang.Long@be991a08
123456789012
123456789012
```



BSF4Rexx

BSF.CLS – New Feature

- Camouflaging Java fields as if they were ooRexx attributes
 - Querying the value of a Java field by merely sending the Java field's name
 - Setting the value of a Java field by merely sending the Java field's name followed by the assignment operator and new value

BSF4Rexx: Accessing Static Fields

BSF.CLS – New Feature

- Sometimes one needs to access static values of Java (interface) classes
- Public routine `bsf.wrapStaticFields()`

```
dir=bsf.wrapStaticFields(nameOfJavaInterfaceClass)
```

```
javaClassName="org.oorexx.datergf.DTC" /* interface class defining constants */
dtc=bsf.wrapStaticFields(javaClassName) /* wrap up interface class */
say "version:" dtc~version "january:" dtc~january

::requires bsf.cls /* load the Java support */
```

Possible Output:

```
version: 92.20060101 january: 1
```



BSF4Rexx – Getting at Event Objects, 1


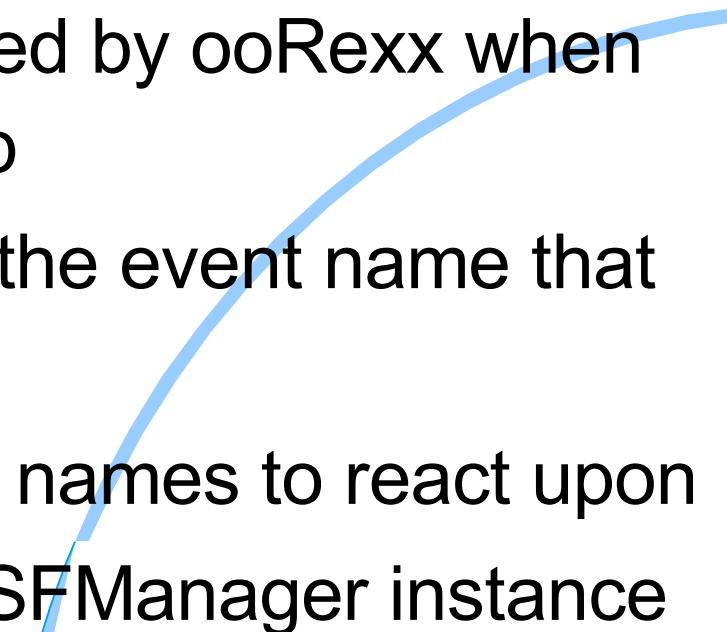
BSF.CLS – New Feature

- Allows retrieving the Java event object giving further information of the event
 - The event object's bean name (index into the BSF registry) will be encoded in the leading comment inserted by BSF4Rexx
- New subfunction, method of BSF.CLS
`bsf.addEventListenerReturningEventInfos()`
- New routine in BSF.CLS
`bsf.getEventInfoObject(eventText)`
 - Returns a proxy (array) object that will remove the event Java object from the BSF registry upon deletion



BSF4Rexx – Getting at Event Objects, 2

BSF.CLS – New Feature

- Information in the received array object `arr`
 - [1] ... an array of the arguments that the event generated, usually the respective event object is at the first index, ie. `arr[1][1]`
 - [2] ... `.nil` or data as supplied by ooRexx when event adapter was set up
 - [3] ... string denominating the event name that has occurred
 - [4] ... `.nil` or string of event names to react upon
 - [5] ... a reference to the BSFManager instance
- 
- 

BSF4Rexx - BSF.Dialog

BSF.CLS – New Feature

- Public class `bsf.dialog`
- Multiplatform, uses Java's swing GUI
- Dialog (class or instance) methods

`.nil=.bsf.dialog~messageBox(message, [title], [type])`

`buttonNumber=.bsf.dialog~dialogBox(message, [title], [type], [optionType], [icon],
[txtButtons], [defaultTxtButton])`

`text=.bsf.dialog~inputBox(message, [title], [type], [icon],
[txtOptions], [defaultTxtOption])`

where "type": error, information, plain, question, warning

Where "optionType": default, OkCancel, YesNo, YesNoCancel

If using the class object (`.BSF.DIALOG`), then the dialog is centered relative to physical screen, if created for a Java window object the dialog is modal for it and centered relative to it.

BSF4Rexx: BSF.Dialog

Examples, 1

```
say "Using class object .BSF.DIALOG, hence centered relative to screen..."
.bsf.dialog~messageBox("Think about it!")

say "dialogBox: returns -1 for escape, 0 for first button, 1 for second button..."
pause
buttonText=.array~of("Save it 0", "Delete 1", "Copy 2", "whoops, that's it!!")
say .bsf.dialog~dialogBox("Please choose one", "Choices", "warning", , , buttonText, "Delete 1")

say "inputBox: returns .nil for escape, text value entered or chosen..."
pause
buttonText=.array~of("Save it 0", "Delete 1", "Copy 2", "whoops, that's it!!")
say .bsf.dialog~inputBox("Please choose one", "Choices", "information", , buttonText, "Delete 1")

say .bsf.dialog~inputBox("Please enter your name:", "Querying stuff", "question")

::requires bsf.cls                               /* load the Java support */
```

Possible Output:

```
Using class object .BSF.DIALOG, hence centered relative to screen...
dialogBox: returns -1 for escape, 0 for first button, 1 for second button...
Drücken Sie eine beliebige Taste . . .
1
inputBox: returns .nil for escape, text value entered or chosen...
Drücken Sie eine beliebige Taste . . .
Delete 1
Rony G. Flatscher
```

BSF4Rexx: BSF.Dialog

Examples, 2 (Relative to a Frame)

```
f=.bsf~new("java.awt.Frame", "Hello!") /* create a Java frame object */
f~~pack ~show
fdlg=.bsf.dialog~new(f) /* create a bsf.dialog instance for this Java frame */

say "Using an instance of .BSF.DIALOG, hence centered relative to a frame object..."
fdlg~messageBox("Think about it!")

say "dialogBox: returns -1 for escape, 0 for first button, 1 for second button..."
pause
buttonText=.array~of("Save it 0", "Delete 1", "Copy 2", "whoops, that's it!!")
say fdlg~dialogBox("Please choose one", "Choices", "warning", , , buttonText, "Delete 1")

say "inputBox: returns .nil for escape, text value entered or chosen..."
pause
buttonText=.array~of("Save it 0", "Delete 1", "Copy 2", "whoops, that's it!!")
say fdlg~inputBox("Please choose one", "Choices", "warning", , , buttonText, "Delete 1")

::requires bsf.cls /* load the Java support */
```

Possible Output:

```
Using an instance of .BSF.DIALOG, hence centered relative to a frame object...
dialogBox: returns -1 for escape, 0 for first button, 1 for second button...
Drücken Sie eine beliebige Taste . . .
3
inputBox: returns .nil for escape, text value entered or chosen...
Drücken Sie eine beliebige Taste . . .
whoops, that's it!!
```

▼ BSF4Rexx – Installation Scripts Running on Linux, Windows

- `setupBSF.rex [path2java.exe [dir4scripts]]`
 - `installBSF4Rexx.{cmd|sh}`
 - `uninstallBSF4Rexx.{cmd|sh}`
- `setupOOo.rex path2OOoSOHomeDir`
 - `installOOo.{cmd|sh}`
 - `uninstallOOo.{cmd|sh}`
- `setupJava.rex`
 - Linux only

▼ BSF4Rexx – Vienna Version Goodies, 1

- Date and time arithmetics/manipulations
- Java version of the datergf package, named
 - org.oorexx.datergf
 - DTC ... defines datergf constants
 - DateRGF
 - e.g. subtractions, additions, determining Easter, Labor Day...
 - TimeRGF
 - DateTimeRGF
 - DateFormatRGF
 - Allows formatting of date and time values with easy to apply formatting patterns

▼ BSF4Rexx – Vienna Version Goodies, 2

- org.oorexx.misc
 - Class `RgfFilter`
 - Implements the Java interface "java.io FilenameFilter"
 - Needed e.g. for file dialogs that need to filter the files to be displayed
- org.rexxla.bsf
 - Class `RexxDispatcher`
 - Allows starting BSF4Rexx Rexx programs from the command line via Java, supplying the command line arguments to the Rexx program

▼ BSF4Rexx – Vienna Version Goodies, 3

- org.oorexx.uno
 - RgfReflectUNO
 - A Java class allowing for full reflection/introspection of UNO objects and/or UNO IDL definitions
 - Results are delivered as strings
- Quite a few new nutshell examples
 - Lee's examples of the 2006 Symposium demonstrating platform independent GUI and printing for ooRexx
 - OpenOffice.org/StarOffice automation examples



Roundup and Outlook

- Vienna Version of BSF4Rexx
 - Introduces typeless interaction with Java
 - Adds utility routines for easing interfacing with Java considerably, e.g.
 - `box()`, `unbox()`, `bsf.wrapStaticFields()`
 - Public routines `iif()`, `pp()`
 - Public class `BSF.Dialog` to allow for using cross-platform `messageBox()`, `dialogBox()`, `inputBox()` functionality

▼ Open Issues

- Real-time handling of events
 - E.g. no canceling possible
- Creating Java proxy objects for Java interfaces
 - E.g. Java Filter interface class
 - At the moment one needs to create a Java class which implements the Java interface and control that from ooRexx
- Creating ooRexx proxy objects to which Java methods can be forward to
 - implementing Java methods in ooRexx

External BSF4Rexx Functions - Overview

The external BSF4Rexx functions allow interfacing with Java. If the Rexx script was invoked by Java, then the external Rexx function `BSF()` is registered already.

BSF Main function to interface with Java.

BSFDropFuncs Drops all registered BSF4Rexx functions.

BSFInvokedBy Returns 0, if no Java is present, returns 1 if BSF4Rexx was invoked by Java, returns 2 if BSF4Rexx was invoked by Rexx.

BSFLoadFuncs Loader function for registering the external BSF4Rexx functions.

BSFLoadJava Loads Java. Optionally pass each startup Java argument as its own argument to this function. If the `-Djava.class.path=...` argument is given, the environment CLASSPATH value will not be used for starting up Java.

BsfQueryALLFunctions Returns a stem array denoting all external BSF4Rexx functions defined in `BSF4Rexx.dll` (OS/2, Windows) resp. `LibBSF4Rexx.so` (Linux, Unix).

BSFQueryRegisteredFunctions Returns a stem array denoting all external BSF4Rexx functions that are registered and can be therefore used.

BSFUnloadJava Unloads Java (has no effect at present).

BSFVersion Returns the version string of the BSF4Rexx dynamic link library, in the form `xnn.yyyymmdd_rexx-engine-package-name`, where `x` denotes the major version number, `nn` the minor version number, and `yyymmdd` the date; after the space the Java package name of the Java BSF Rexx engine is given.

Loading the External BSF4Rexx Functions

```
if rxFuncQuery("BSF") = 1 then /* not registered yet, hence load it! */
do
  call rxFuncAdd "BsfLoadFuncs", "BSF4Rexx", "BsfLoadFuncs"
  call BsfLoadFuncs /* registers all BSF4Rexx functions */
  call BSFLoadJava /* loads Java */
end
```

Subfunctions of BSF()

The external Rexx function `BSF()` is the main interface to Java from Rexx and resides in the BSF4Rexx DLL/so. As such this function offers a wealth of functionality, organized into subfunctions, e.g.:

```
call BSF "sleep", 1.05 /* sleep 1050 msec, invoked as a procedure */
```

or

```
res=BSF("sleep", 1.05) /* sleep 1050 msec, invoked as a function */
```

Sometimes return values or arguments indicate that no value is supplied. For that purpose one uses the string `".NIL"`, which represents the Java value `null`.

In very rare cases it is necessary to indicate the exact type of an argument. All subfunction names containing the string `strict` expect a typeIndicator to precede each argument, according to the following table (bold characters in typeIndicator strings must be at least given):

TypeIndicator	The Immediately Following Argument is of Type
"B0olean"	boolean , ie. the value 0 (false) or 1 (true)
"BYte"	a byte value
"Char"	char , ie. a single (UTF8) character
"Double"	a double value
"Float"	a float value
"Int"	int , ie. an integer value
"Long"	long a long value
"Object"	a Java object stored in the BSF registry
"SHort"	a short value
"SString"	a String value (UTF8)

In the following subfunctions `beanName` denotes the key in the BSF registry referencing the desired Java object (the bean):

- `addEventListener, beanName, eventSetName, eventName|, eventText`
- `addEventListenerReturningEventInfos, beanName, eventSetName, - eventName|, eventText, sendBackData`
- `arrayAt, arrayBeanName, idx0 [, idx1]... arrayAt, arrayBeanName, intArrayBean`
- `arrayLength beanName`
- `arrayPut, arrayBeanName, newValue, idx0 [, idx1]... arrayPut, arrayBeanName, newValue, intArrayBean`
- `arrayPutStrict, arrayBeanName, typeIndicator, newValue, idx0 - [, idx1]... arrayPutStrict arrayBeanName, typeIndicator, newValue, intArrayBean`
- `createArray, JavaClassObjectBeanName, dim0 [, dim1]... createArray, JavaClassObjectBeanName, intArrayBean`
- `exit [, [exitCode] [, time2wait_in_msec]]`
- `getFieldValue, beanName, fieldName getFieldValueStrict, beanName, fieldName`
- `getPropertyValue, beanName, propertyName, index|.NIL`
- `getStaticValue, JavaClassName, fieldName getStaticValueStrict, JavaClassName, fieldName`
- `invoke, beanName, methodName [, argument]... invokeStrict, beanName, methodName [, typeIndicator, argument]...`
- `loadClass, JavaClassName`
- `lookupBean, beanName`
- `pollEventText [, timeout_in_msec]`
- `postEventText, eventText[, priority] (priority: 0=low,1=normal,2=high)`
- `registerBean|new [beanName], JavaClassName [, argument]...`
- `registerBeanStrict|newStrict, [beanName], JavaClassName - [, typeIndicator, argument]...`
- `setFieldValue, beanName, fieldName, newValue setFieldValueStrict, beanName, fieldName, [typeIndicator,] newValue`
- `setPropertyValue, beanName, propertyName, index|.NIL, - newValue setPropertyValueStrict, beanName, propertyName, index|.NIL, - typeIndicator, newValue`
- `setRexxNullString, newString`
- `sleep, time2sleep_in_seconds`
- `unregisterBean, beanName`
- `version`
- `wrapArray, arrayBeanName`
- `wrapEnumeration, enumerationBeanName`

Preregistered Java Objects (BSF Registry)

To ease creating Java array objects, the most important Java class objects are preregistered in the BSF registry on the Java side. *Please note:* the name of the BSF registry keys for class objects representing the primitive datatypes `byte`, `char`, `short`, `int`, `long`, `float` and `double` start with a *lower case letter*:

BeanName (Key for Registry)	References the Java Class Object
"Array.class"	java.lang.reflect.Array
"Class.class"	java.lang.Class
"Method.class"	java.lang.reflect.Method
"Object.class"	java.lang.Object
"String.class"	java.lang.String
"System.class"	java.lang.System
"Thread.class"	java.lang.Thread
"boolean.class"	boolean (primitive datatype)
"Boolean.class"	java.lang.Boolean
"byte.class"	byte (primitive datatype)
"Byte.class"	java.lang.Byte
"char.class"	char (primitive datatype)
"Character.class"	java.lang.Character
"double.class"	double (primitive datatype)
"Double.class"	java.lang.Double
"float.class"	float (primitive datatype)
"Float.class"	java.lang.Float
"int.class"	int (primitive datatype)
"Integer.class"	java.lang.Integer
"long.class"	long (primitive datatype)
"Long.class"	java.lang.Long
"short.class"	short (primitive datatype)
"Short.class"	java.lang.Short
"void.class"	void (primitive datatype)
"Void.class"	java.lang.Void



ooRexx Interface (Module BSF.CLS)

The object-oriented interface support for ooRexx is realized by calling or requiring the ooRexx module `BSF.CLS`, which defines public routines, classes and the environment symbol `.BSF4REXX` (a directory containing BSF objects). You can get at that support in one of two ways:

```
call BSF.CLS /* make oo-like BSF4Rexx support available */
or
::requires BSF.CLS /* make oo-like BSF4Rexx support available */
```

Sometimes return values or arguments indicate that no value is supplied. For that purpose the ooRexx object `.nil` (an environment symbol, hence not quoted) is used, which represents the Java value `null`.

Most of the BSF4Rexx subfunctions are made available as class or instance methods of the public class `BSF`, prepended with the string `"bsf."`.

The public class `BSF` is used for representing Java (class) objects. Its instances are proxy objects which forward received messages to the Java side for invocation. When such ooRexx proxy objects get garbage collected, they will cause the BSF registry reference counter of the represented Java object to be decreased. If that reference counter drops to zero, the Java object gets removed from the BSF registry.

Public Routines

- `box(typeIndicator, value)` wraps `value` as a Java object of type `typeIndicator` and returns the reference to it
- `bsf.createArray(...)` see BSF-subfunction
- `bsf.getConstant(JavaClassName, fieldName)`
- `bsf.getEventInfoObject(eventText)`
- `bsf.getStaticValue(...)` see BSF-subfunction
- `bsf.getStaticValueStrict(...)` see BSF-subfunction
- `bsf.import(javaClassName[, .nil|name4.local])`
- `bsf.loadClass(...)` see BSF-subfunction
- `bsf.lookupBean(...)` see BSF-subfunction
- `bsf.pollEventText(...)` see BSF-subfunction
- `bsf.postEventText(...)` see BSF-subfunction
- `bsf.unregisterBean(...)` see BSF-subfunction
- `bsf.wrap(strBSFRegistryKey)` returns an ooRexx proxy object, if string is a Java object in the BSF registry, otherwise the supplied argument
- `bsf.wrapStaticFields(JavaClassName)` returns an ooRexx directory object
- `iif(truthValue, valueIfTrue, valueIfFalse)` returns `valueIfTrue`, if first argument is `.true`, returns `valueIfFalse` else
- `pp(argument)` returns argument's string value enclosed in square parenthesis
- `unbox(o)` returns the value from Java's primitive datatype wrapper object `o`

Public Class BSF

This is the ooRexx proxy class for representing Java classes. ooRexx messages sent to it or its instances cause the invocation of the appropriate Java methods. All methods starting with `bsf.` are pass-through methods and its arguments are documented in the BSF subfunctions on the other page.

If the ooRexx proxy object gets garbage collected by ooRexx, the reference counter for the represented Java object in the BSF registry gets decreased (if that counter hits zero, the Java object is removed from the BSF registry, such that it can also be garbage collected by Java).

BSF's CLASS METHODS

- `bsf.createArray(...)` see BSF-subfunction
- `bsf.exit(...)` see BSF-subfunction
- `bsf.getStaticValue(...)` see BSF-subfunction
- `bsf.getStaticValueStrict(...)` see BSF-subfunction
- `bsf.import(JavaClassName [, .nil | name4.local])`
- `bsf.loadClass(JavaClassName)` see BSF-subfunction
- `bsf.lookupBean(...)` see BSF-subfunction
- `bsf.pollEventText(...)` see BSF-subfunction
- `bsf.postEventText(...)` see BSF-subfunction
- `bsf.setRexxNullString(...)` see BSF-subfunction
- `bsf.sleep(...)` see BSF-subfunction
- `bsf.wrapArray(...)` see BSF-subfunction
- `bsf.wrapEnumeration(...)` see BSF-subfunction
- `newStrict([typeIndicator, argument]...)` only available, if the ooRexx class object proxy was created with `bsf.import()`; creates an instance of the given Java class using typed arguments (as opposed to the `new` method) and returns an ooRexx proxy object

BSF's INSTANCE METHODS

- `bsf.class` returns Java class object proxy
- `bsf.addEventListener(...)` see BSF-subfunction
- `bsf.addEventListenerReturningEventInfos(...)` see BSF-subfunction
- `bsf.exit(...)` see BSF-subfunction
- `bsf.invoke(...)` see BSF-subfunction
- `bsf.invokeStrict(...)` see BSF-subfunction
- `bsf.getFieldValue(...)` see BSF-subfunction
- `bsf.getFieldValueStrict(...)` see BSF-subfunction
- `bsf.setFieldValue(...)` see BSF-subfunction
- `bsf.setFieldValueStrict(...)` see BSF-subfunction
- `bsf.getPropertyValue(...)` see BSF-subfunction
- `bsf.setPropertyValue(...)` see BSF-subfunction
- `bsf.setPropertyValueStrict(...)` see BSF-subfunction

Private Class BSF_ARRAY_REFERENCE

`BSF_ARRAY_REFERENCE` is a subclass of `BSF` that allows interacting with Java array objects (stored in the BSF registry) as if they were ooRexx arrays (e.g. index values start with `1`, and the ooRexx array methods `AT`, `[]`, `DIMENSION`, `ITEMS`, `MAKEARRAY`, `PUT`, `[]=`, `SUPPLIER` are implemented).

The public routine `bsf.wrap` will use this class to create the ooRexx proxy object, if it detects that the supplied BSF registry key refers to a Java array object.

Public Class BSF_PROXY

`BSF_PROXY` is a subclass of `BSF` that allows creating proxy objects by passing the BSF registry key (a string) to its init method. This allows sending ooRexx messages which will cause the appropriate Java methods to be invoked.

Unlike direct instances of the `BSF` class, these proxy objects, if garbage collected, do not decrease the BSF registry counter.

Directory Object .BSF4Rexx

`BSF.CLS` will initialize a directory object to contain proxies to all preregistered Java objects in the BSF registry. As ooRexx will

translate statements into uppercase before executing them, the directory index values `Byte.class` and `byte.class` cannot be distinguished anymore as they both become `BYTE.CLASS`. Therefore the index value of the primitive class object references do not carry the suffix `.CLASS`.

Get the Java version by querying the Java class object `java.lang.System`:

```
call bsf.cls /* load the ooRexx support for BSF4Rexx */
say .bsf4rex--system.class -getProperty('java.version')
```

or create a two dimensional Java array (3 by 4 elements) of the primitive datatype `int`:

```
a=bsf.createArray(.bsf4rex--int, 3, 4) /* create Java int array (3x4) */
::requires BSF.CLS /* load the ooRexx support for BSF4Rexx */
```

Index	References the Java Class Object
ARRAY.CLASS	java.lang.reflect.Array
CLASS.CLASS	java.lang.Class
METHOD.CLASS	java.lang.reflect.Method
OBJECT.CLASS	java.lang.Object
STRING.CLASS	java.lang.String
SYSTEM.CLASS	java.lang.System
THREAD.CLASS	java.lang.Thread
BOOLEAN	boolean (primitive datatype)
BOOLEAN.CLASS	java.lang.Boolean
BYTE	byte (primitive datatype)
BYTE.CLASS	java.lang.Byte
CHAR	char (primitive datatype)
CHARACTER.CLASS	java.lang.Character
DOUBLE	double (primitive datatype)
DOUBLE.CLASS	java.lang.Double
FLOAT	float (primitive datatype)
FLOAT.CLASS	java.lang.Float
INT	int (primitive datatype)
INTEGER.CLASS	java.lang.Integer
LONG	long (primitive datatype)
LONG.CLASS	java.lang.Long
SHORT	short (primitive datatype)
SHORT.CLASS	java.lang.Short
VOID	void (primitive datatype)
VOID.CLASS	java.lang.Void

Public Class BSF.DIALOG

Supplies the methods `messageBox()`, `dialogBox()` or `inputBox()`, via the class object `.BSF.DIALOG` or an instance of it.

- `messageBox(message, [title], [type])` always returns `.nil`
- `dialogBox(message, [title], [type], [optionType], [icon], [txtButtons], [defaultTxtButton])` returns button number (0=first button)
- `inputBox(message, [title], [type], [icon], [txtOptions], [defaultTxtOption])` returns entered/chosen text

where:

`type` one of `error`, `information`, `plain`, `question`, `warning`
`optionType` one of `default`, `OkCancel`, `YesNo`, `YesNoCancel`

```
.bsf.dialog--messageBox("Think about it!") /* using the class object */
/* create and show a frame object for which a modal dialog is used */
f=.bsf-new("java.awt.Frame", "Hello!")~pack~show /* create and show */
fdlg=.bsf.dialog--new(f) /* create an instance, tell it about 'f' */
say fdlg--dialogBox("Continue?", "warning", "YesNo") /* modal to 'f' */
```