An Introduction to the Bean Scripting Framework

Victor J. Orlikowski
orlikowski@apache.org

ApacheCon US - Las Vegas
November 20, 2002

What is BSF?

The Bean Scripting Framework (BSF) is...

What is BSF?

The Bean Scripting Framework (BSF) is...

- a system by which JSPs may be implemented in languages other than Java (i.e. Javascript, Python, or Tcl)

What is BSF?

The Bean Scripting Framework (BSF) is...

- a system by which JSPs may be implemented in languages other than Java (i.e. Javascript, Python, or Tcl)
- a system by which servlets (or any Java application) may incorporate scripting languages for implementation or extension

What is BSF?

The Bean Scripting Framework (BSF) is...

- a system by which JSPs may be implemented in languages other than Java (i.e. Javascript, Python, or Tcl)
- a system by which servlets (or any Java application) may incorporate scripting languages for implementation or extension
- a system allowing Java objects and functions to be used from within scripting languages

What is BSF?

The Bean Scripting Framework (BSF) is...

- a system by which JSPs may be implemented in languages other than Java (i.e. Javascript, Python, or Tcl)
- a system by which servlets (or any Java application) may incorporate scripting languages for implementation or extension
- a system allowing Java objects and functions to be used from within scripting languages

BSF accomplishes this by providing an API for Java to call scripting languages to execute scripts and an object registry that exposes Java objects (rather than only Java Beans) to scripting languages.

A Little History

- Began as a research project at IBM's T.J. Watson Research Labs

A Little History

- Began as a research project at IBM's T.J. Watson Research Labs
- Posted on AlphaWorks website in June of 1999;
  Moved to DeveloperWorks thereafter.

A Little History

- Began as a research project at IBM's T.J. Watson Research Labs
- Posted on AlphaWorks website in June of 1999;
  Moved to DeveloperWorks thereafter.
- Released to Apache from IBM in 2002; now a subproject of Jakarta

A Little History

- Began as a research project at IBM's T.J. Watson Research Labs
- Posted on AlphaWorks website in June of 1999;
  Moved to DeveloperWorks thereafter.
- Released to Apache from IBM in 2002; now a subproject of Jakarta

Authors

- Original authors:
  - Sanjiva Weerawarana
  - Matthew Duftler
  - Sam Ruby

A Little History (cont.)

Authors (cont.)

- Authors of initial pass at debugging
  - Olivier Gruber
  - Jason Crawford
  - John Ponzo
- Author of several modifications to Jasper involving BSF
  - John Shin
- Current maintainers (also had a hand in adding debugging)
  - Chuck Murcko
  - Victor Orlikowski

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

- Javascript (via Rhino)
- Python (via Jython or JPython)
- Tcl (via Jacl)
- NetRexx
- XSLT stylesheets (via Xalan and Xerces)

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

- Javascript (via Rhino)
- Python (via Jython or JPython)
- Tcl (via Jacl)
- NetRexx
- XSLT stylesheets (via Xalan and Xerces)

Languages that are likely to soon be added are:

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

- Javascript (via Rhino)
- Python (via Jython or JPython)
- Tcl (via Jacl)
- NetRexx
- XSLT stylesheets (via Xalan and Xerces)

Languages that are likely to soon be added are:

- Ruby (via JRuby)
- BeanShell
- JudoScript
- Perl

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

- Javascript (via Rhino)
- Python (via Jython or JPython)
- Tcl (via Jacl)
- NetRexx
- XSLT stylesheets (via Xalan and Xerces)

Languages that are likely to soon be added are:

- Ruby (via JRuby)
- BeanShell
- JudoScript
- Perl

Languages that are not actively maintained:

Supported Languages

In its current version (2.3), BSF supports the following scripting languages:

- Javascript (via Rhino)
- Python (via Jython or JPython)
- Tcl (via Jacl)
- NetRexx
- XSLT stylesheets (via Xalan and Xerces)

Languages that are likely to soon be added are:

- Ruby (via JRuby)
- BeanShell
- JudoScript
- Perl

Languages that are not actively maintained:

- VBScript, LotusScript and PerlScript were supported under an ActiveScript engine, but this engine is not currently actively developed.

Architecture

BSF has two main architectural features:

Architecture

BSF has two main architectural features:

- the BSFManager (provides scripting services to Java apps, and maintains the object registry)

Architecture

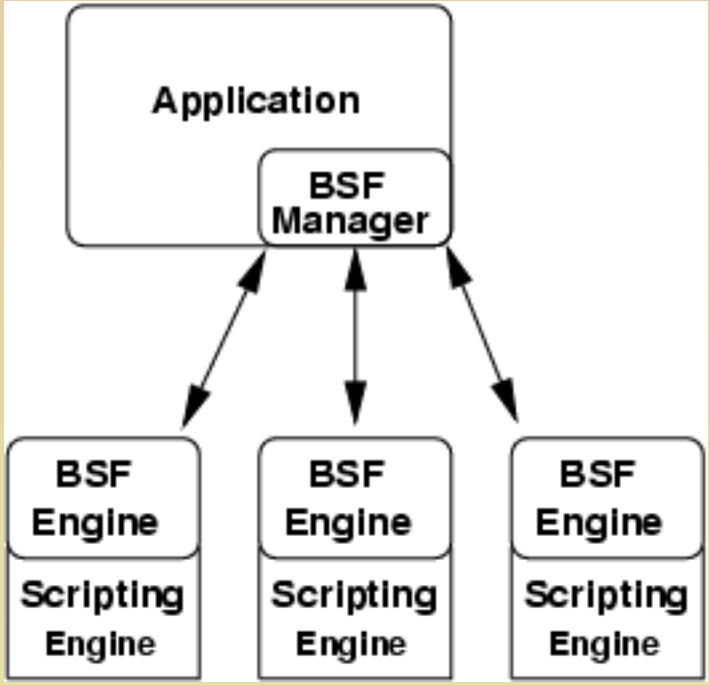
BSF has two main architectural features:

- the BSFManager (provides scripting services to Java apps, and maintains the object registry)
- the BSFEngine (provides an abstraction over the scripting engine, which is what actually executes scripts on behalf of the app)

Architecture

BSF has two main architectural features:

- the BSFManager (provides scripting services to Java apps, and maintains the object registry)
- the BSFEngine (provides an abstraction over the scripting engine, which is what actually executes scripts on behalf of the app)



Application Server Installation

Application Server Installation

Tomcat setup and standalone use

- Packages, patches and instructions for Tomcat support are available on BSF's site on Jakarta (http://jakarta.apache.org/bsf/index.html)

Application Server Installation

Tomcat setup and standalone use

- Packages, patches and instructions for Tomcat support are available on BSF's site on Jakarta (http://jakarta.apache.org/bsf/index.html)

IBM Websphere Application Server

- Version 5.0 includes BSF 2.3; versions 3.0 and 4.0 include earlier versions

Application Server Installation

Tomcat setup and standalone use

- Packages, patches and instructions for Tomcat support are available on BSF's site on Jakarta (http://jakarta.apache.org/bsf/index.html)

IBM Websphere Application Server

- Version 5.0 includes BSF 2.3; versions 3.0 and 4.0 include earlier versions

When developing standalone Java applications, ensure that bsf.jar and any jars required for your desired scripting language are in the CLASSPATH.
When executing scripts calling Java functions or using Java objects, BSF provides a wrapper for calling the scripts in bsf.jar

Usage - JSP

Under either Tomcat or Websphere, set the language attribute
in the page directive to make a JSP use a language other than Java.

```
<%@ page language=‘‘javascript’’ %>
```

This instructs the page compiler to call BSF for any scriptlets, expressions, etc.
in the JSP, and informs BSF that the language being used is Javascript.

Usage - JSP (cont.)

JSPs using scripting languages under BSF have the same set of JSP implicit objects available as Java JSPs.

In fact, input and output must be handled via these implicit objects, since the scripting language engines are not aware of the fact that they are being called from within a JSP.

Usage - JSP (cont.)

JSPs using scripting languages under BSF have the same set of JSP implicit objects available as Java JSPs.

In fact, input and output must be handled via these implicit objects, since the scripting language engines are not aware of the fact that they are being called from within a JSP.

Example:

```
<html>
<head><title>A Minimal Hello World JSP</title></head>
<%@ page language=‘‘javascript’’ %>
<body>
<%
    out.println(‘‘<b>Hello world, from Javascript!</b>’’);
%>
</body>
</html>
```

Usage - JSP (cont.)

JSPs using scripting languages under BSF have the same set of JSP implicit objects available as Java JSPs.

In fact, input and output must be handled via these implicit objects, since the scripting language engines are not aware of the fact that they are being called from within a JSP.

Example:

```
<html>
<head><title>A Minimal Hello World JSP</title></head>
<%@ page language=‘‘javascript’’ %>
<body>
<%
    out.println(‘‘<b>Hello world, from Javascript!</b>’’);
%>
</body>
</html>
```

In this case, BSF is used to execute the scriptlet, and uses the `out` implicit object to print a boldface string.

Usage - JSP (cont.)

JSPs using BSF are not limited to using a single language.
Available from the Jakarta taglibs project,
the BSF taglibs provide the following two tags:

Usage - JSP (cont.)

JSPs using BSF are not limited to using a single language.
Available from the Jakarta taglibs project,
the BSF taglibs provide the following two tags:

- scriptlet

Usage - JSP (cont.)

JSPs using BSF are not limited to using a single language.
Available from the Jakarta taglibs project,
the BSF taglibs provide the following two tags:

- scriptlet
- expression

Usage - JSP (cont.)

JSPs using BSF are not limited to using a single language.
Available from the Jakarta taglibs project,
the BSF taglibs provide the following two tags:

- scriptlet
- expression

Both of these tags have a required language attribute, which allows you to specify the language used on a per-expression or per-scriptlet basis.

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

- Create a new BSFManager

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

- Create a new BSFManager
- Register or declare any desired objects

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

- Create a new BSFManager
- Register or declare any desired objects
- Load any desired scripting engines

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

- Create a new BSFManager
- Register or declare any desired objects
- Load any desired scripting engines
- Execute/evaluate scripts

Usage - Servlets and Applications

When writing servlets or applications using BSF, the following steps must be followed (after importing the BSF classes):

- Create a new BSFManager
- Register or declare any desired objects
- Load any desired scripting engines
- Execute/evaluate scripts

The following code snippet illustrates the process:

Usage - Servlets and Applications (cont.)

```
import com.ibm.bsf.*;

class Foo {
    BSFManager mgr = new BSFManager ();
    ...
    public void fooExec() {
        mgr.registerBean(‘‘myWin’’, myWindow);
        BSFEngine jsEngine =
              mgr.loadScriptingEngine(‘‘javascript’’);
        jsEngine.exec(‘‘fooScript.js’’, 0, 0, ‘‘2+2’’);
    }
}
```

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

- `public BSFManager()`
  BSFManager constructor

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

- `public BSFManager()`
  BSFManager constructor

- `public Object eval(String lang, String source,`
  `int lineNo, int columnNo, Object expr)`
  Mirrors BSFEngine eval() method - evaluates an expression, and returns the resulting value, after loading the appropriate scripting engine, based on the language `lang`

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

- `public BSFManager()`
  BSFManager constructor

- `public Object eval(String lang, String source,`
  `int lineNo, int columnNo, Object expr)`
  Mirrors BSFEngine eval() method - evaluates an expression, and returns the resulting value, after loading the appropriate scripting engine, based on the language `lang`

- `public void exec(String lang, String source,`
  `int lineNo, int columnNo, Object script)`
  Mirrors BSFEngine exec() method - executes a script, after loading the appropriate scripting engine

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

- `public BSFManager()`
  BSFManager constructor

- `public Object eval(String lang, String source,`
  `int lineNo, int columnNo, Object expr)`
  Mirrors BSFEngine eval() method - evaluates an expression, and returns the resulting value, after loading the appropriate scripting engine, based on the language `lang`

- `public void exec(String lang, String source,`
  `int lineNo, int columnNo, Object script)`
  Mirrors BSFEngine exec() method - executes a script, after loading the appropriate scripting engine

- `public BSFEngine loadScriptingEngine(String lang)`
  Loads and returns a BSFEngine matching a scripting engine of the desired type

Usage - Servlets and Applications (cont.)

Most commonly used BSFManager methods:

- `public BSFManager()`
  BSFManager constructor

- `public Object eval(String lang, String source, int lineNo, int columnNo, Object expr)`
  Mirrors BSFEngine eval() method - evaluates an expression, and returns the resulting value, after loading the appropriate scripting engine, based on the language `lang`

- `public void exec(String lang, String source, int lineNo, int columnNo, Object script)`
  Mirrors BSFEngine exec() method - executes a script, after loading the appropriate scripting engine

- `public BSFEngine loadScriptingEngine(String lang)`
  Loads and returns a BSFEngine matching a scripting engine of the desired type

The BSFManager also provides methods for compiling scripts and expressions into a CodeBuffer object, and for calling anonymous methods.

Usage - Servlets and Applications (cont.)

Three other particularly useful BSFManager methods are:

Usage - Servlets and Applications (cont.)

Three other particularly useful BSFManager methods are:

- `public void registerBean(String beanName, Object bean)`
  Used to register an object with the BSF object registry, and thereby make it available to scripts, when looked up using `lookupBean()`

Usage - Servlets and Applications (cont.)

Three other particularly useful BSFManager methods are:

- `public void registerBean(String beanName, Object bean)`
  Used to register an object with the BSF object registry, and thereby make it available to scripts, when looked up using `lookupBean()`

- `public Object lookupBean(String beanName)`
  Used to look up objects in the object registry, and return them for use in a script

Usage - Servlets and Applications (cont.)

Three other particularly useful BSFManager methods are:

- `public void registerBean(String beanName, Object bean)`
  Used to register an object with the BSF object registry, and thereby make it available to scripts, when looked up using `lookupBean()`

- `public Object lookupBean(String beanName)`
  Used to look up objects in the object registry, and return them for use in a script

- `public void declareBean(String beanName, Object bean, Class type)`
  Used to make an object "automatically" available to scripts (not needing to be looked up using `lookupBean()`

Usage - Servlets and Applications (cont.)

Three other particularly useful BSFManager methods are:

- `public void registerBean(String beanName, Object bean)`
  Used to register an object with the BSF object registry, and thereby make it available to scripts, when looked up using `lookupBean()`
- `public Object lookupBean(String beanName)`
  Used to look up objects in the object registry, and return them for use in a script
- `public void declareBean(String beanName, Object bean, Class type)`
  Used to make an object "automatically" available to scripts (not needing to be looked up using `lookupBean()`

The first two methods provide access to Java objects via BSF's object registry. The third makes the specified object an implicit object within the context of all currently loaded scripting language engines.

Usage - Servlets and Applications (cont.)

Within the execution context of any script that is being executed underneath BSF, an implicit `bsf` object is available, which represents the BSFManager that is managing the script.

This object provides access to all of the BSFManager's methods, but is usually used for calling `lookupBean()` within scripts in order to access objects made available through the object registry.

Usage - Servlets and Applications (cont.)

Two most commonly used BSFEngine methods:

Usage - Servlets and Applications (cont.)

Two most commonly used BSFEngine methods:

- `public Object eval(String source, int lineNo, int columnNo, Object script)`
  Evaluate an expression, and return the value

Usage - Servlets and Applications (cont.)

Two most commonly used BSFEngine methods:

- `public Object eval(String source, int lineNo, int columnNo, Object script)`
  Evaluate an expression, and return the value
- `public void exec(String source, int lineNo, int columnNo, Object script)`
  Execute a script

Usage - Servlets and Applications (cont.)

Two most commonly used BSFEngine methods:

- `public Object eval(String source, int lineNo, int columnNo, Object script)`
  Evaluate an expression, and return the value
- `public void exec(String source, int lineNo, int columnNo, Object script)`
  Execute a script

In order to add a new language engine, one must implement the BSFEngine interface for the new language.

Incorporating your own scripting language

If you have written a scripting language, and would like to hook it into a Java application using BSF, you must do the following:

Incorporating your own scripting language

If you have written a scripting language, and would like to hook it into a Java application using BSF, you must do the following:

- Implement the BSFEngine interface for your language

Incorporating your own scripting language

If you have written a scripting language, and would like to hook it into a Java application using BSF, you must do the following:

- Implement the BSFEngine interface for your language
- Create an instance of BSFManager in your application

Incorporating your own scripting language

If you have written a scripting language, and would like to hook it into a Java application using BSF, you must do the following:

- Implement the BSFEngine interface for your language
- Create an instance of BSFManager in your application
- Make the BSFManager aware of your language via the `registerScriptingEngine()` method.

Incorporating your own scripting language

If you have written a scripting language, and would like to hook it into a Java application using BSF, you must do the following:

- Implement the BSFEngine interface for your language
- Create an instance of BSFManager in your application
- Make the BSFManager aware of your language via the `registerScriptingEngine()` method.

```
public static void registerScriptingEngine (String lang,
                String engineClassName, String[] extensions)
```

`lang` is the language name for use with the `loadScriptingEngine()` method, `engineClassName` is the name of your class implementing BSFEngine, and `extensions` is an array of possible filename extensions for your scripts.

Running standalone scripts

BSF provides a wrapper for running standalone scripts that require it.

Simply run:

```
java com.ibm.bsf.Main
```

and a help message will be provided.

As in the case of all other scripts, the implicit `bsf` object is available to standalone scripts.

Debugging (Javascript only)

Over the last year, debugging capabilities have been added to BSF. However, there are several limitations.

Debugging (Javascript only)

Over the last year, debugging capabilities have been added to BSF. However, there are several limitations.

- Debugging is limited to Javascript only

Debugging (Javascript only)

Over the last year, debugging capabilities have been added to BSF. However, there are several limitations.

- Debugging is limited to Javascript only
- Debugging as currently designed is intended for use only with JSPs

Debugging (Javascript only)

Over the last year, debugging capabilities have been added to BSF. However, there are several limitations.

- Debugging is limited to Javascript only
- Debugging as currently designed is intended for use only with JSPs
- Only a rudimentary command-line debugger is available freely (IBM has a product based on the Eclipse IDE that includes a debugging plugin for BSF)

Debugging (Javascript only)

Over the last year, debugging capabilities have been added to BSF.
However, there are several limitations.

- Debugging is limited to Javascript only
- Debugging as currently designed is intended for use only with JSPs
- Only a rudimentary command-line debugger is available freely (IBM has a product based on the Eclipse IDE that includes a debugging plugin for BSF)

An emphasis was placed on designing a generic API for script debugging;
this is still an area in need of work.
Interested parties are encouraged to contribute.

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Thanks go out to:

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Thanks go out to:

- IBM Corporation, for sponsoring BSF and donating it to Apache

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Thanks go out to:

- IBM Corporation, for sponsoring BSF and donating it to Apache
- Apache Software Foundation, for accepting BSF as a project

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Thanks go out to:

- IBM Corporation, for sponsoring BSF and donating it to Apache
- Apache Software Foundation, for accepting BSF as a project
- All of BSF's authors

Links and Thanks

The latest version of these slides can always be downloaded from:
http://www.dulug.duke.edu/~vjo/papers/ApacheCon_US_2002/

Thanks go out to:

- IBM Corporation, for sponsoring BSF and donating it to Apache
- Apache Software Foundation, for accepting BSF as a project
- All of BSF's authors
- Chuck Murcko and Bill Stoddard, for reviewing initial drafts of these materials