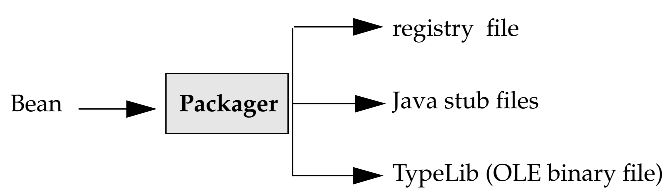
Using JavaBeans with Microsoft ActiveX Components

JavaBeans may be used as embedded objects or Microsoft ActiveX components within applications such as Microsoft Office, Internet Explorer, and Visual Basic. This document provides an overview of the Bridge and instructions on how to package and register Beans as ActiveX components.

Overview

To use a Bean as an embedded object, you package the Bean and then use it as an ActiveX component.



A registry file (text file) is created by the packager as an interim step. This registry contains an object id, the executable path for the component, bridge information, and a type library path.

The Packager also generates Java stub files (both .java source and compiled .class files). These stub files constitute a bridge between the Bean and ActiveX.

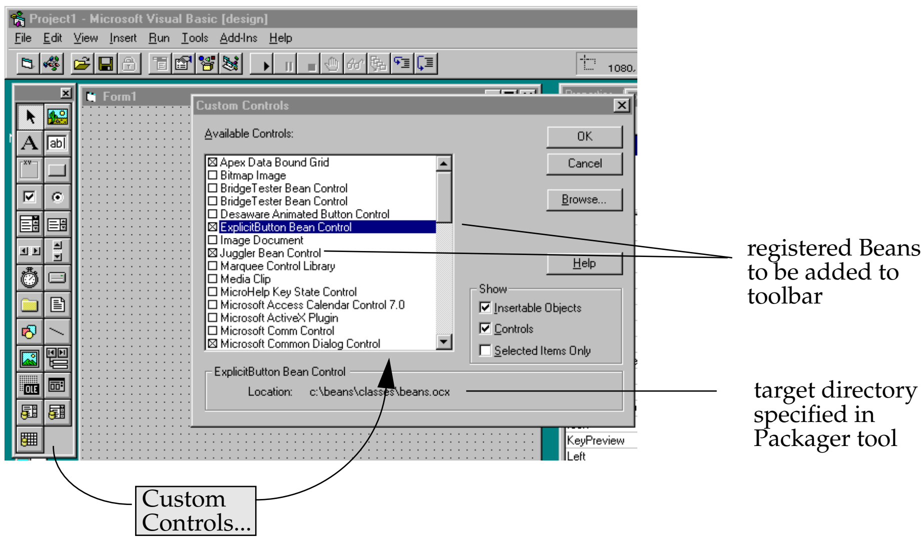
The TypeLib file is a binary file which describes each component's properties, events, and methods. The sources are deleted upon completion of the packaging step, and the .class files are added to the jar file originally containing the JavaBean. This jar file remains completely portable if the `sun.beans.ole` classes are available when you try to load the jar. If you want to use these jar files with an environment that doesn't use the ActiveX bridge or did not install the java classes, package and distribute a copy of the jar file.

Using Beans as ActiveX Components

The following example uses the Microsoft Visual Basic development environment to illustrate how two sample Beans, ExplicitButton and Juggler, may be used as ActiveX components. For convenience, these beans were registered as ActiveX components at installation time.

Importing Beans into Visual Basic

From within Visual Basic, select the "Tools" menu and "Custom Controls" item and add the ExplicitButtonBeanControl and JugglerBeanControl as shown below.

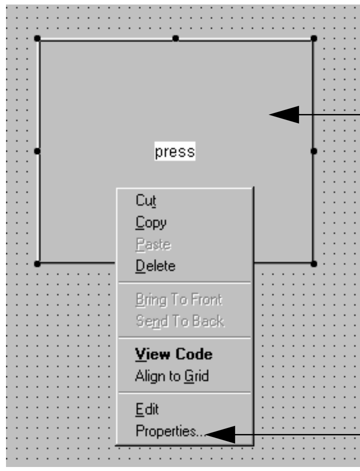


After importing the custom controls, Juggler and ExplicitButton will appear on the toolbar.

Example

You can work with Juggler and ExplicitButton in the same way as you would with ordinary ActiveX components. For example, instantiate a Juggler and two ExplicitButtons in the Visual Basic form. Label the ExplicitButtons “Start” and “Stop” in the Properties window (by editing the “Label” property).

Alternatively, you can invoke the customizer on the Bean (if the bean has a customizer) by right-clicking on the component and selecting the Properties menu as shown below

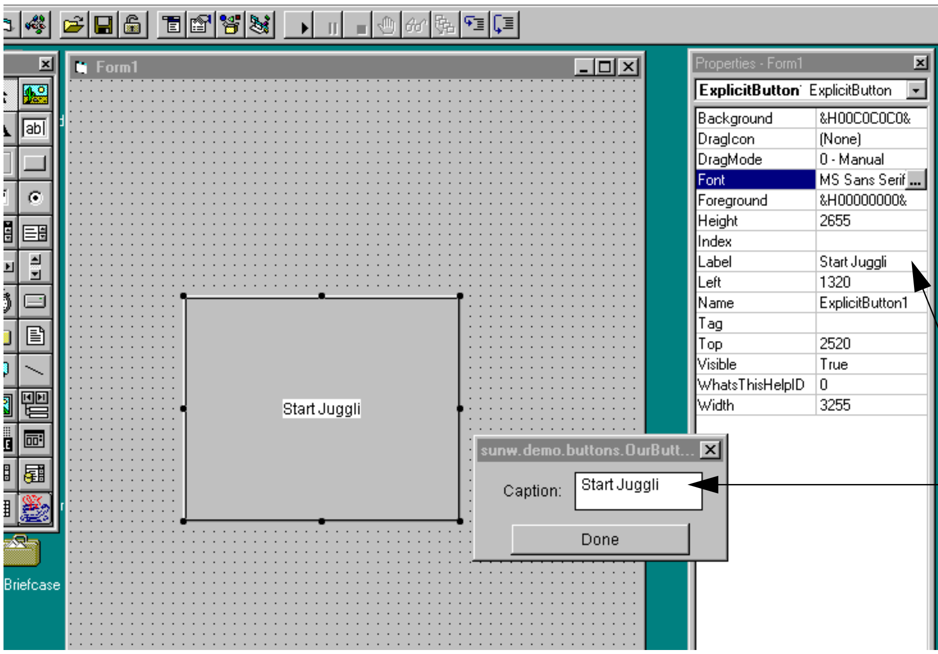


Right Click on the Bean in design-mode in Visual Basic

Select the Properties...

Once the customizer is open, modify the label of the button. Notice that the label changes in the Visual Basic properties tool bar. This occurs because the button fires a PropertyChangeEvent which is caught by the bridge and forwarded to the container (in this case Visual Basic).

The customizer is also available through the normal PropertyPages mechanism of the ActiveX specifications. Please refer to your ActiveX container documentation on how to open the property pages of an ActiveX component.



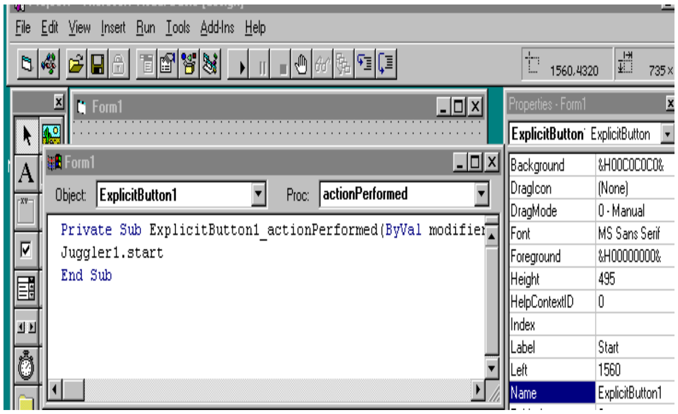
Notice that the text stays in sync when you type a new label.

Double click on the "Start" button and add the event handling Visual Basic code :

Juggler1 is the "Name" property of the Juggler instance

```
Juggler1.start
```

to the `ExplicitButton1_actionPerformed` subroutine as illustrated below.

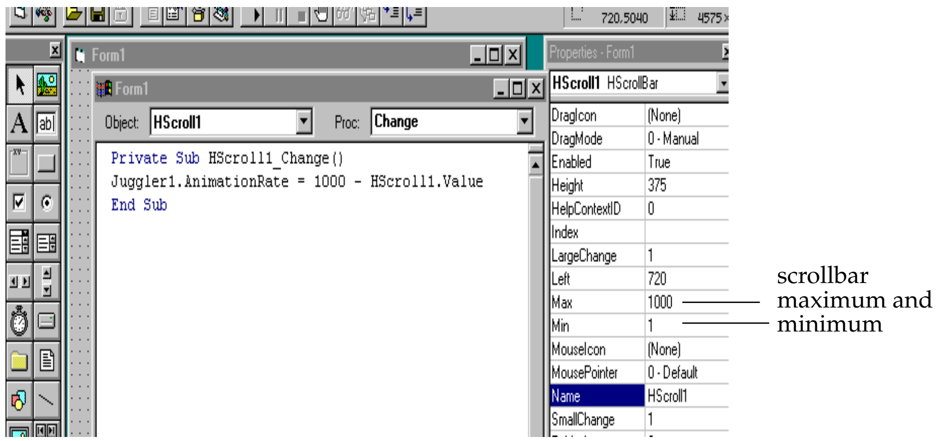


Now, double-click on the “Stop” button and add the event handling Visual Basic code :

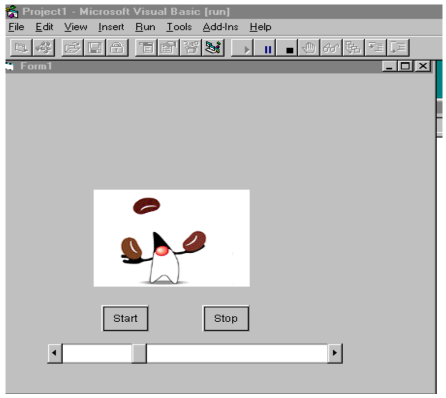
```
Juggler1.stop
```

to the `ExplicitButton2_actionPerformed` subroutine. Run the program and test your work by pressing the “Start” and “Stop” buttons to start and stop the Juggler.

Add an ActiveX scrollbar component and use it to control the Juggler's animation speed. Double-click on the scrollbar to edit the `HScroll1_Change` subroutine as shown below.



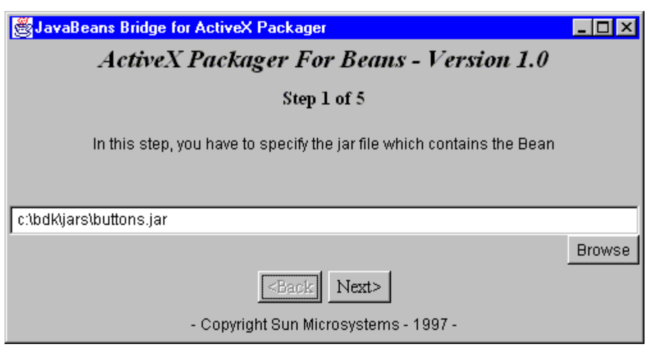
Run the program. You can now increase and decrease the Juggler's animation speed by moving the scrollbar right and left.



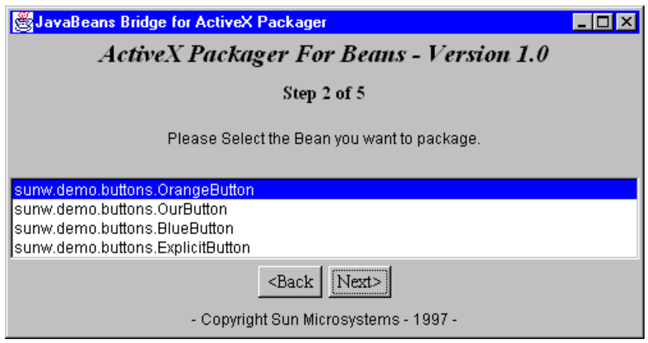
Packaging and Registration

1. **Start the Packager.**
   `Open the Windows Start Menu and select the ActiveX Packager For Java Beans icon from the Programs group.`

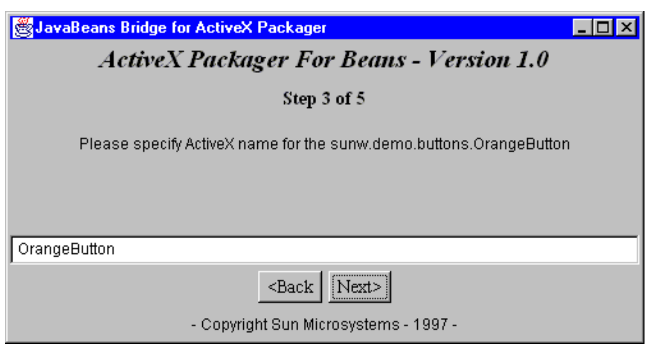
2. **Specify the Jar File**
   The packaging step will guide you through five steps to package your Bean. For the first step, specify which jar file contains the Bean to be packaged and press "Next"



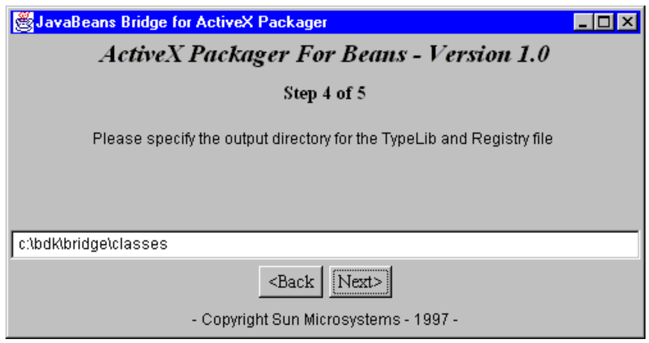
3. **Select the JavaBean**
   For the second step, the packaging tool lists all the Beans inside the jar file you specified in step 1. Select the Bean to be packaged and press "Next".



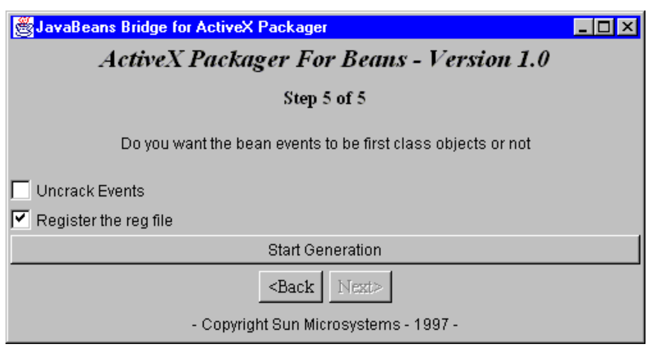
4. **Specify an ActiveX name**
   For the third step, the packaging tool suggests an ActiveX name for the Bean. This name will be used in the ActiveX environment to refer to the component. The packager gives you a default name (the name of the Bean minus its java package name prefix).



5. **Specify an output directory**
   The packager will create a registry file and a type library in a directory that you specify. Note that the registry file contains information on the location of the type library file. If you move the type library file (.tlb), you will need to manually update the registry file and register it again. It is assumed that the `beans.ocx` file is also located in this output directory.



6. **Start Generation**
   For this step, you must choose between the cracked or uncracked events mechanism of event generation (see “Event object or not” on page 18).



Command Line options

The packager can be invoked from the command line with the following arguments :

```
java sun.beans.ole.Packager
   -jar      <jar file path>
   -n        <Bean name as referenced in the jar manifest file>
   -ax       <ActiveX name to be associated with the bean>
   -o        <output directory>
   -awt      <generate events as AWT events>
```

You may specify options in any order. You must use the -jar option; the packager will use the following default values for options which you do not specify.

- `-n`
  The packager uses the first Bean available in the jar file. This option is recommended when the specified jar file contains more than one Bean.
- `-ax`
  The packager uses the Bean name (minus the package name prefix). For example, `sunw.demo.buttons.ExplicitButton` becomes `ExplicitButton`.
- `-o`
  The packager generates the typelib and reg file in the current directory. Specify your desired directory.
- `-awt`
  Specify to generate OLE events following the AWT standard.

The packager displays status messages in the standard output of the command prompt.

JavaBeans Bridge Run-Time Support

To make packaged JavaBeans easily distributed with your Windows application, a run-time version of the JavaBeans Bridge for ActiveX is provided. By bundling the packaged JavaBeans and the Bridge Runtime, any Windows application which takes advantages of JavaBeans can be deployed easily.

The bridge installation includes a JavaBeans Bridge Run-Time setup to install the run-time part of the bridge (without the packager or demo registration) on a client machine. Find it in the <rtsetup> subdirectory of your installation. This setup is a standard InstallShield installation procedure.

Minimum Requirement for distributing packaged JavaBeans

When a packaged JavaBeans is distributed across machines (or Internet), the following requirement must be met in order for the packaged JavaBeans to work:

- Java Development Kit (JDK) / Java RunTime Environment (JRE) 1.1 or above must be installed in the target machine.
- JavaBeans Bridge for ActiveX Runtime 1.0 or JavaBeans Bridge for ActiveX 1.0 must be installed as well.
- Each packaged JavaBeans consists of at least a JAR file, a TLB (Type Library) file and a REG (Registry) file. They must be reachable from the target machine, and the settings in the REG file should be imported into the target machine's registry. Note: REG file contains the path of the JAR file and TLB file. If you move the JAR and TLB files in different locations in different machines, make sure the corresponding registry settings are updated accordingly.

Internet Explorer Support

HTML Design time

Starting with version 3.0, Microsoft Internet Explorer support ActiveX components that can be downloaded and run within an HTML page.

The bridge can be used to run JavaBeans components inside IE 3.0 using all the features of JDK 1.1 or higher. To use the bridge in Intenet Explorer, the bridge can be used like it is used in Visual Basic :

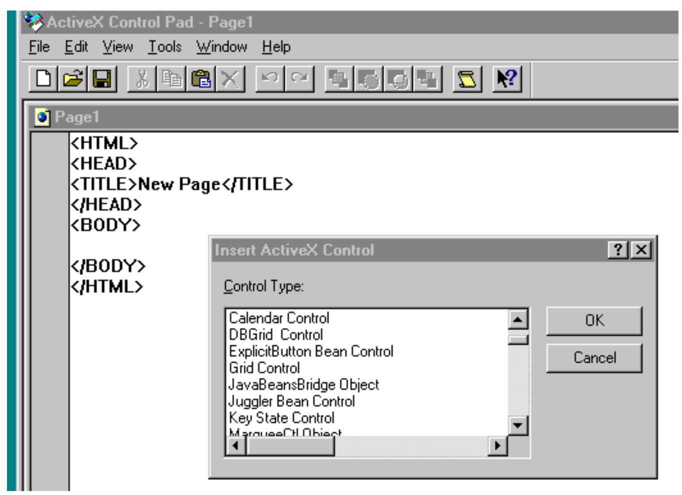
- Package the JavaBean
- Paint it using an HTML authoring tool like FrontPage 97 or notepad
- Run it inside Internet Explorer

The JavaBean component (jar file, type lib and registration) will have to be installed on each target machine using either a custom installation or technologies like CAB files (see CAB SDK from Microsoft).

Alternatively, the bridge is able to support non registered JavaBeans components on a target machine if the information about the bean is stored in the properties of the ActiveX control (using the PARAM tags). For that the bridge is using a unique CLSID, that allow it to run in any ActiveX container application but cannot run any particular JavaBean, let's call it the generic CLSID of the bridge.

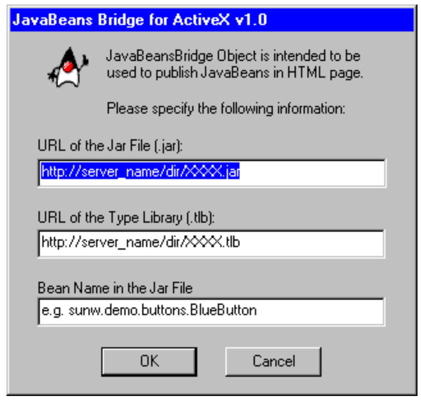
Let's take an example on how one could paint an ExplicitButton on an HTML page using Microsof ActiveX Control Pad. First the bean has to be packaged on the developer machine, maybe not registered but the typelib and the modified jar file must be available.

From the Edit menu of Control Pad, choose the "Insert an ActiveX control..." option and the following screen should appear :



As you can see, the ExplicitButton is registered on this machine, but we are going to select the JavaBeansBridge.Object control. This is our generic ActiveX bridge control to which you need to specify which bean component you want to run.

Once the JavaBeansBridge.Object is selected, the JavaBeans bridge is loaded and the bridge will display a dialog box requesting the information about which JavaBean component you are looking to run :

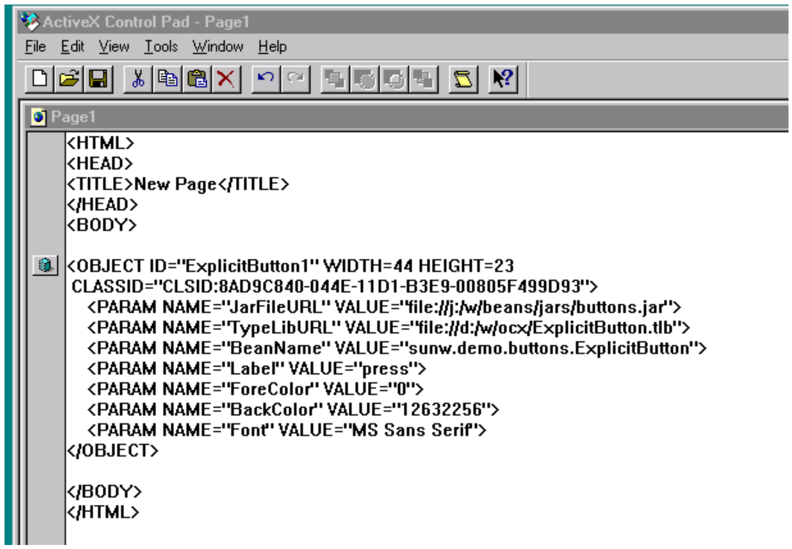


Note that the information requested is in URL format so if your jar file is stored locally in the file system, use “file://c:/bdk/bridge/jars/buttons.jar”.

Once the information about the localization of the packaged jar file, the type lib file and the bean name is entered, click Ok and the button will appear in the design-window.

The properties does not appear at design-time in the property sheet because ControlPad and other Microsoft Container like FrontPage and Visual Basic requires the TypeLib to be available without loading the ActiveX control. As a consequence, you cannot use the JavaBeansBridge.Object inside Visual Basic, you need to use registered beans as explained in Chapter 1.

Close the design-window and the control will be persisted to an HTML page like



You can now modify the textual properties if you want to like changing the label from “press” to “Press Me !”.

You can also attach scripting methods inside this html page :

```
<Script LANGUAGE=”VBScript”>
<!---
Sub ExplicitButton1_actionPerformed(ActionEvent1)
       Alert “I have been clicked”
end sub
--->
</SCRIPT>
```

Any other Active enabled authoring tool like FrontPage 97 should provide the same experience.

Run-time support

In this version, we suppose the JavaBeans Bridge for ActiveX has been installed on the client machine. Once this is done, the user can display the HTML page containing the “Bridged” JavaBean component. No registration is necessary on the client machine. The bridge will download the jar file and the type lib into Internet Explorer disk cache and will run the JavaBean.

Once the page is not accessed any more, the cache will be purged by Internet Explorer and the client system will be clean as if he had never run this JavaBean component.

ActiveX Bridge Technical Details

The ActiveX run-time is implemented using the COM threading model. When you run several JavaBeans inside a multi-threaded ActiveX container (currently only Internet Explorer), each JavaBean is likely to run in a separate thread. If your Beans are making direct connections or sharing objects it's probably a good idea to think about concurrent access.

Design Time environment

The Packager application and the packager.dll are needed at design time only. Once the JavaBeans component has been successfully packaged into an ActiveX component, these classes and dll will not be used.

Be sure to use the ActiveX for JavaBeans Packager icon from the Programs folder in your windows taskbar. This ensures that your `CLASSPATH` and `PATH` are set correctly. .

Run Time environment

At run-time, the ActiveX bridge does not need any particular `CLASSPATH` or PATH settings; the Registry is used to get all necesary information the bridge might need.

The registry keys used by the bridge are :

`HKEY_LOCAL_MACHINE`

`Software\\JavaSoft\\Java Beans\\ActiveX Bridge\\ActiveXHome`

This key refers to the installation directory of the bridge where it should find the .class it needs. If you choose not to set this key, you must set the `CLASSPATH`.

`Software\\JavaSoft\\Java Runtime\\CurrentVersion\\JavaHome`

or

```
SOFTWARE\\JavaSoft\\Java Development Kit\\1.1.x\\JavaHome
```

One of these two keys must be defined or the JDK must be accessible through the `PATH`.

Persistent Storage

In order to run successfully on the ActiveX bridge, a JavaBean must be serializable through either serialization or externalization. The current interfaces supported by the bridge are IPersistStorage, IPersistStreamInit, and IPersistPropertyBag.

When a Bean is asked to serialize itself, it must perform all serialization in one thread.

The bridge may choose to persist the bridge through textual property persistence. All public properties will be persisted though the values the accessor methods return. If a particular property is a Java Object, the PropertyEditor getAsText/setAsText methods will be used. If they are not implemented, serialization will be use on the property.

As stated for Code Generation in the JavaBeans specification version 1.01, The bean designer may specify that a bean cannot be restored by simply saving and restoring its properties. This can be done by using the Feature Descripto's attribute/value mechanism (see JavaBeans spec § 5.4.1).

If the bean descriptor property "hidden-state" is set to true, the bean will always be persisted through serialization or externalization.

Events

All source interfaces that are declared as default in the EventSetDescriptor array returned by the BeanInfo will be merged into one interface which is declared as being the ActiveX default source interface. It is an ActiveX requirement that at least the default source interface is accessible by ActiveX containers.

Each method returned by the getListenerMethodDescriptors API of the EventSetDescriptor is mapped to an ActiveX event. The event name is the method name.

Consider, for example the `java.awt.event.ActionListener` source interface :

```
public interface ActionListener extends EventListener {
    /**
     * Invoked when an action occurs.
     */
    public void actionPerformed(ActionEvent e);
}
```

The ActiveX event name generated from this interface definition is actionPerformed.

Since all default interfaces of your Bean are merged to a unique interface, do not use interfaces that contain the same method name.

Event object or not

The new AWT event model specifies that all event notifications come with a unique event object containing all information describing the event. Although this is required, the event object fields should be immutable.

This is different from how an ActiveX programmer receives events in OLE. Usually, ActiveX events are delivered with several native types that describe the event. Some of these fields are read-only, others may be read-write.

The ActiveX bridge gives you the opportunity at packaging time to decide how you want to deliver the events to the ActiveX world. If you choose to uncrack the event, all of the event properties will be extracted on the fly and the OLE event signature will be the list of all the event properties. Alternatively, you can use the AWT model; the event will be delivered with a unique IDispatch parameter that you can use to request information about the event.

Consider again the `java.awt.event.ActionListener` interface. The following illustrates how the event is generated if you choose to crack the event :

```
Private Sub ExplicitButton1_actionPerformed(
    ByVal modifiers As Long,
    ByVal actionCommand As String,
    ByVal ID As Long,
    ByVal source As Object)
     Place Code here....
End Sub
```

If you choose to not crack the event, the event signature is

```
Private Sub ExplicitButton1_actionPerformed
    (ByVal ActionEvent1 As Object)
End-Sub
```

The packager tool default is to uncrack events. You can set this option in step 5 of the packager tool.

Properties

All properties are accessible though the bridge with the same access right as defined in the PropertyDescriptor. The ActiveX bridge will automatically invoke the accessor methods defined in the PropertyDescriptor.

If the property is bound or constrained, the [bindable] or [requestedit] flags are set in the description of the OLE property. The IPropertyNotifySink interface will forward the PropertyChangeEvent. If the OLE container denies the property change and the property is constrained, the PropertyVetoException is thrown by the bridge.

Native properties will be mapped to their OLE type as described below

Table 5-1 Java to Automation Mappings

Java Type	**Automation Type**
boolean	VT_BOOL
char	VT_UI1
double	VT_R4
float	VT_R2
int	VT_I4
byte	VT_I2
short	VT_I2

Table 5-1 Java to Automation Mappings

Java Type	Automation Type
long	VT_I4 (truncated)
java.lang.String	VT_BSTR
java.awt.Color	VT_COLOR
java.awt.Font	VT_FONT
Arrays (1 dimension)	VT_ARRAY | type
All others	VT_DISPATCH

All property names will have their first letter recapitalized for consistency with ActiveX conventions (label becomes Label).

For simple types and string, font and color most ActiveX containers like Visual Basic will allow direct manipulation of the properties in their PropertySheet. For all other properties like Arrays or Objects, the properties will not be accessible in the PropertySheet but can still be invoked using the scripting language offered by the container.

Methods

All methods declared in the BeanInfo via a MethodDescriptor will be automable from OLE Automation capable containers. Overloaded methods are currently not accepted by automation; the bridge will use the methods with the most arguments as the preferred.

Arguments and return values are automatically marshalled by the bridge (refer to Table 5-1 for translation values). Exceptions thrown by the method will be caught by the bridge, packaged in the EXCEPINFO structure and returned to the OLE container.

Ambient Properties

The bridge supports four ambient properties that get translated to the JavaBeans properties if the JavaBean component implements these properties. These ambient properties will be read and set on the JavaBeans component when the component is first painted on the canvas (InitNew) and each time a change notification for the ambient properties is received.

Properties that map standard control properties are shown below.

Table 5-2 Standard DISPID Assignment

Java Type and Name	Dispatch DISPID
java.awt.Color , background	DISPID_BACKCOLOR
java.awt.Color, foreground	DISPID_FORECOLOR
java.awt.Font, font	DISPID_FONT
boolean, enable	DISPID_ENABLED

ActiveX container ambient property changes of the previous types will be forwarded by the bridge to the bean via a property set on the property.

Introspection

Introspection is not used at run-time, only at packaging time. All methods and properties will be accessible as described earlier. With release v1.0, only the source interfaces that declared as being part of the default set will be packaged as a unique ActiveX source interface.

Customization

Customizers, when defined, are available at design-time in Visual Basic though the Property Pages of the component or by invoking the menu item Properties by right clicking on the JavaBeans.

Packaging

The ActiveX bridge supports Beans packaged in jar files only. The packager will add stub .class file in this jar file. You can still use the jar file once packaged with the beanbox application on any platform if the classpath includes the bridge java classes (sun.beans.ole.*).

Registry File

The packager will generate the type library file (repository of the capabilities of the Bean) and the registry file. The registry indicates the executable location, the type library location, and other information. You may edit the registry file. Register the registry file as follows:

```
regedit FileName.reg
```

Information contained in the registry file :

Executable location: this is where you have copied the beans.ocx file. One `beans.ocx` is enough to run all the packaged beans. You don't need to duplicate this .ocx file for each Bean.

```
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\InprocServer32]
@= "c:\\perso\\beans\\beans.ocx
```

The registry file has at least three lines of information as shown below:

- Jar file location where to load the Bean from,
- name of the Bean inside the jar file.
- name of the class responsible for interfacing the ActiveX native code and the Bean. You can change it but this is only recommended for only for experienced Java programmers. The interface of the class has to be the same.

```
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\JarFileName]
@= "g:\\workspaces\\beans\\jars\\buttons.jar"
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\JavaClass]
@= "sunw.demo.buttons.ExplicitButton"
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\InterfaceClass]
@= "sun/beans/ole/OleBeanInterface"
```

Here is the location of the bitmap and the icon for the ocx. Use a gif->ico or gif->bmp translator program to move your gifs to windows specific format and put the path for these files here. If you want to load them from an executable, follow the example below where 1 is the ResourceID forthe bitmap.

```
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\ToolboxBitmap32]
@= "g:\\workspaces\\bridge\\classes\\beans.ocx,1"
[HKEY_CLASSES_ROOT\CLSID\{23BE13F0-4B64-11CB-BB1D-00805F2ADE08}\DefaultIcon]
@= "g:\\workspaces\\bridge\\classes\\hotjava.ico"
```

Here is the location of the type library generated by the packaging step. Change this information only if you move files around.

```
[HKEY_CLASSES_ROOT\TypeLib\{23BE13F1-4B64-11CB-BB1D-00805F2ADE08}]
@= "ExplicitButton Bean Control Type Library"
[HKEY_CLASSES_ROOT\TypeLib\{23BE13F1-4B64-11CB-BB1D-00805F2ADE08}\1.0]
@= "ExplicitButton Bean Control "
[HKEY_CLASSES_ROOT\TypeLib\{23BE13F1-4B64-11CB-BB1D-00805F2ADE08}\1.0\0\win32]
@= "g:\\workspaces\\bridge\\classes\\ExplicitButton.tlb"
[HKEY_CLASSES_ROOT\TypeLib\{23BE13F1-4B64-11CB-BB1D-00805F2ADE08}\1.0\FLAGS]
@= "2"
[HKEY_CLASSES_ROOT\TypeLib\{23BE13F1-4B64-11CB-BB1D-00805F2ADE08}\1.0\HELPDIR]
@= "g:\\workspaces\\bridge\\classe
```

Automation Advanced Usage

Arrays manipulation

Indexed properties and other arrays are accessible through the bridge as SafeArrays. Arrays of all types (except Java long) can be obtained or passed to a JavaBeans component from any OLE compliant scripting environment. Here is an example of how to retrieve a array of short from a hypothetical JavaBeans property called ShortArray and display all its values in a Visual Basic Dialog Box :

```
Rem // Declare the array variable
Dim myArray as Variant
NL = Chr(10);
Rem // Do a property get on my JavaBeans component
myArray = MyJavaBeansComponent.ShortArray
// Construct the message box text
Msg = "Array from " & LBound(myArray) & " to " & UBound(myArray)
For j = LBound(myArray) to UBound(myArray)
   Msg = Msg & NL & "Element " & j & " = " & myArray(j)
Next
Rem // Display the message Box
MsgBox Msg
```

Indexed properties can also be set using an index and a value or by setting the entire array in one property set. Let's examine examples of the two methods:

```
Rem // Set the 4th element of the array to the value 36
MyJavaBeansComponent.setShortArrayWithIndex(2, 36)

Rem // Set the entire indexed property
Dim myArray(3)
Dim var as Variant
x(0) = 9
x(1) = 18
x(2) = 36
var = myArray
MyJavaBeansComponent.ShortArray(var)
```

Object creation

Two methods allow you to create new Java objects from any scripting facility. These methods are accessible from all Java objects visible in the scripting environment but not the Bean itself. This means that any object that a Bean returns as a result of a method invocation (for example a property get or event fire) supports these methods.

The first method, getNew, takes one argument of type string that identifies the class name of the object you want to instantiate. The GetNew implementation will try to load the class using the system class loader and instantiate it using the newInstance method of the Class class.

Here is an example in Visual Basic of creating a new Button inside an actionPerformed event listener method implementation :

```
Private Sub button1_actionPerformed(ByVal ActionEvent1 as Object)
    Dim newButton as Object
    Rem // Invoke the getNew on the Event object
    Rem // note that the button1 does not support the the getNew
    set newButton = ActionEvent1.getNew("java.awt.Button");
    MsgBox newButton.toString()
End Sub
```

The second method for creating new Java objects is getConstructor and is provided as part of the JDK1.1 reflection APIs. Using the reflection APIs, one can obtain the list of constructors for a particular class and then can invoke one of these constructors using the Method.Invoke API. getConstructor takes two arguments of type string; the first argument is the class of the object, the second argument is a semi-colon (;) delimited list of parameters. Refer to the

JDK 1.1 reflection API documentation for a complete reference. The bridge will try to find a constructor that satisfies the following pattern :

```
public arg1(arg2);
```

where arg1 and arg2 are the arguments of the getConstructor method. For example, in the java.awt.Button class definition, you will find the constructor :

```
public Button(java.lang.String label);
```

To get this constructor returned by the bridge: (from Visual Basic)

```
Private Sub button1_actionPerformed(ByVal ActionEvent1 as Object)
    Dim constructor as Object
    Rem // Invoke the getConstructor on the Event object
    set constructor = ActionEvent1.getConstructor
            ("java.awt.Button", "java.lang.String");
End Sub
```

The object returned by the getConstructor method is either null if the constructor cannot be found following the design pattern described above or an IDispatch reference on the Method object (see java.lang.reflect.Method). You can use this method object to create a one to many instance of the class you need by invoking the newInstance method on it. If you look in the class java.lang.reflect.Constructor, you'll see that the newInstance method takes an array of Object as the list of parameters to be passed to the constructor. Here is an complete example of how to invoke the Button constructor described earlier:

```
Private Sub button1_actionPerformed(ByVal ActionEvent1 as Object)
    Dim constructor as Object
    Rem // Invoke the getCosntructor on the Event object
    set constructor = ActionEvent1.getConstructor
            ("java.awt.Button", "java.lang.String");

    Rem // Now we have the constructor, let's build the array of
    Rem // arguments used by the constructor (only 1 of type string)
    Dim parms(1)
    parms(0) = " My New Button Label"

    // Let's try to invoke the constructor
    Dim newButton as Object
    set newObject = constructor.newInstance(parms)

    // Let's display something with it
    MsgBox newObject.toString()

End Sub
```

Direct connections between JavaBeans

It is possible to make direct connections between Beans running in the bridge inside the same application. This requires a script for the initial hookup but once accomplished, the two components communicate directly without going through the bridge or COM.

This will allowa JavaBeans component to directly receive events of other JavaBeans components running in the same container without paying the price of the bridge and the COM layers.

Consider a Bean that implements the java.awt.event.ActionListener interface to receive button events. To do so, the Bean can directly implement the listener interface or much more cleanly provide an adaptor class that implements the listener interface. The example below illustrates the latter approach using a new feature of JDK1.1 - nested classes.

```
Class JavaBeansExample {
   .... Some Code....
/*
 * Small adaptor class that implement the ActionListener
 * @return ActionListener interface implementation
 */
public java.awt.event.ActionListener getActionListener() {
   return new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent e) {
        Dialog myDialog = new Dialog(new Frame(),
             "Event Received");
        myDialog.reshape(200,200,400,400);
        myDialog.show();
      }
   };
}
.... Some Code....
```

Invoking the getActionListener method on this Bean returns an object which implements the ActionListener interface. The interface implementation actionPerformed method creates a new Dialog box to notify the user that the ActionEvent has been received.

The ExplicitButton Bean which is delivered as part of the Beans Development Kit sources the ActionListener interface. You can link the interface implementation of the first JavaBean component to an ExplicitButton instance of get the Dialog box to pop up when the user presses the button. The hookup can be done when the Visual Basic form is loaded:

```
Private Sub Form_Load()
   ExplicitButton1.addActionListener(JavaBeans1.getActionListener)
End Sub
```

If the JavaBeansExample component directly implements java.awt.event.ActionListener, the code would look like :

```
Class JavaBeansExample implements java.awt.event.ActionListener {
    ... Some Code ...
    public void actionPerformed(java.awt.event.ActionEvent e) {
          Dialog myDialog = new Dialog(new Frame(),
               "Event Received");
          myDialog.reshape(200,200,400,400);
          myDialog.show();
    }
... Some code...
}
```

And the method hookup would be :

```
Private Sub Form_Load()
    ExplicitButton1.addActionListener(JavaBeans1.Object)
End Sub
```

Either approach is valid. When the user presses the JavaBeans ExplicitButton, the event is caught by the JavaBean Example component which then displays a Dialog Box. These direct connections are not limited to events, an object created by one JavaBean component can be passed as a method parameter to another Bean. Note that the original object is passed and not a copy.

OLE Automation Server

It is possible to take a JavaBeans and make it an OLE Automation Server (no UI). The JavaBeans Bridge for ActiveX has built-in support for this feature, so no change is needed in the JavaBeans.

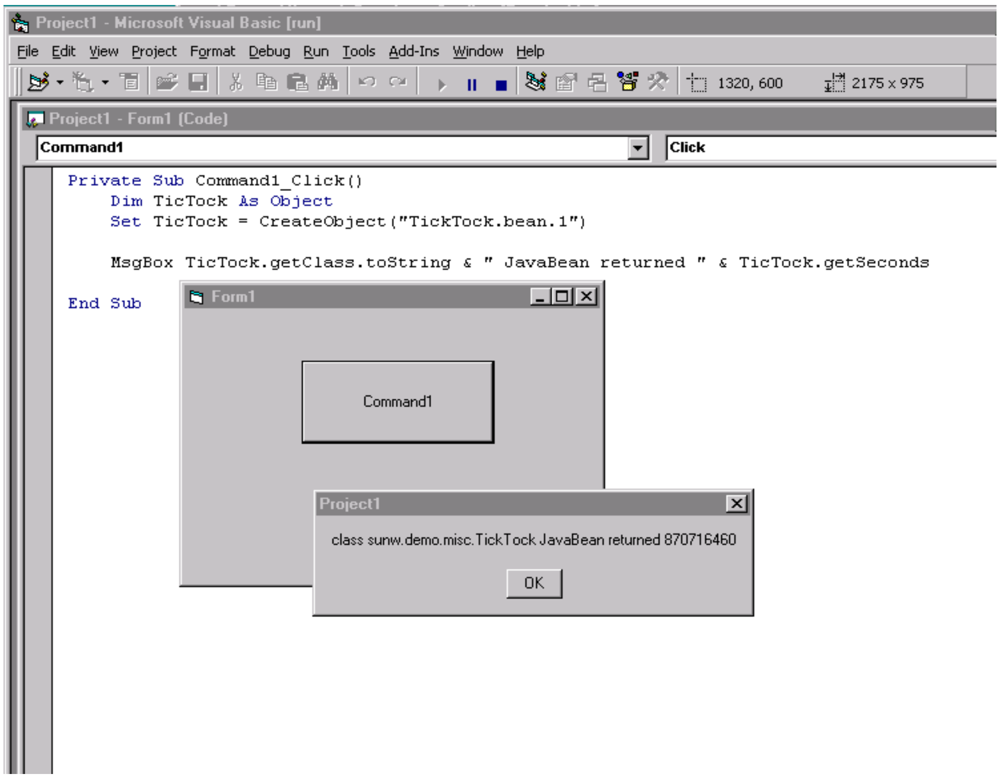
Here is an example in Visual Basic of creating a new TickTock and invoking its methods through OLE Automation:

```
Rem // Load the JavaBeans as an OLE Automation Server.
Rem // "TickTock.Bean.1" is the ProgID of the JavaBean component
Dim TickTock As Object
Set TickTock = CreateObject("TickTock.Bean.1")

Rem // Invoke method on TickTock
MsgBox TickTock.getClass.toString
       & " JavaBean returned " & TickTock.getSeconds

Rem // Release the object
Set TickTock = Nothing
```

The result will be the following :



Unregister JavaBeans over ActiveX

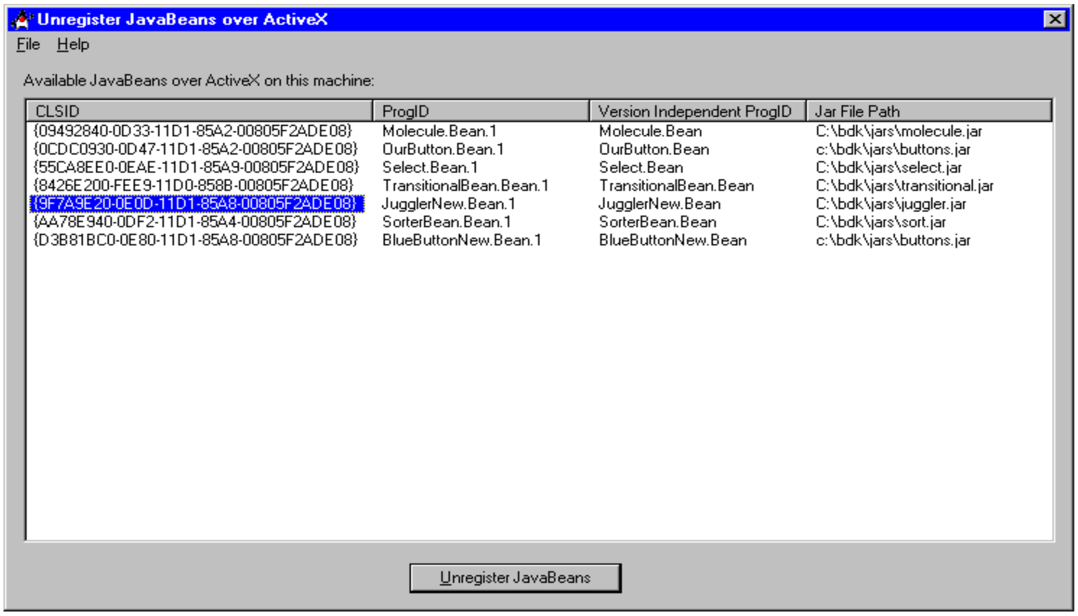
In some cases, unregister a packaged JavaBeans over ActiveX is necessary, so we provide two utilities for performing this task in DOS and Win32. UnregBean is a DOS based program that can be executed from a command prompt. It takes the ProgID of the JavaBeans to perform the unregistration. During JavaBeans packaging time, you are asked for an ActiveX name. The ProgID is normally the ActiveX name + ".Bean.1". For example, if the ActiveX name is Select, the ProgID should be "Select.Bean.1".

```
UnregBean
   -h
   -d
   -i <ProgID>
   -u <ProgID>
```

- `-h`
  The UnregBean displays help message.
- `-d`
  The UnregBean displays information about all the JavaBeans over ActiveX in the current machine. Since the list may be extensive big, we suggest to use "unregbean | more".
- `-i`
  The UnregBean displays information about a specific JavaBeans over ActiveX in the current machine using the ProgID.
- `-u`
  The UnregBean unregisters a specific JavaBeans over ActiveX in the current machine using the ProgID.

The UnregBean displays output messages in the standard output of the command prompt.

WUnregBean is a Win32-based utility for unregistration.



CLSID, ProgID, Version-Independent ProgID and the path of the jar file are displayed. To unregister a JavaBeans over ActiveX, simply clicking the “Unregister JavaBeans” button, and the utility will perform the task.

To remove the stubs in the JAR file during unregistration, UnregBean and WUnregBean require JDK 1.1 to be installed in the target machine. However, if JDK is not installed, UnregBean and WUnregBean will still unregister the selected JavaBeans, but stubs will not be removed from the JAR file.