How to build a bean for Servoy
– A step by step tutorial brought to you by Servoy Stuff

PART 3

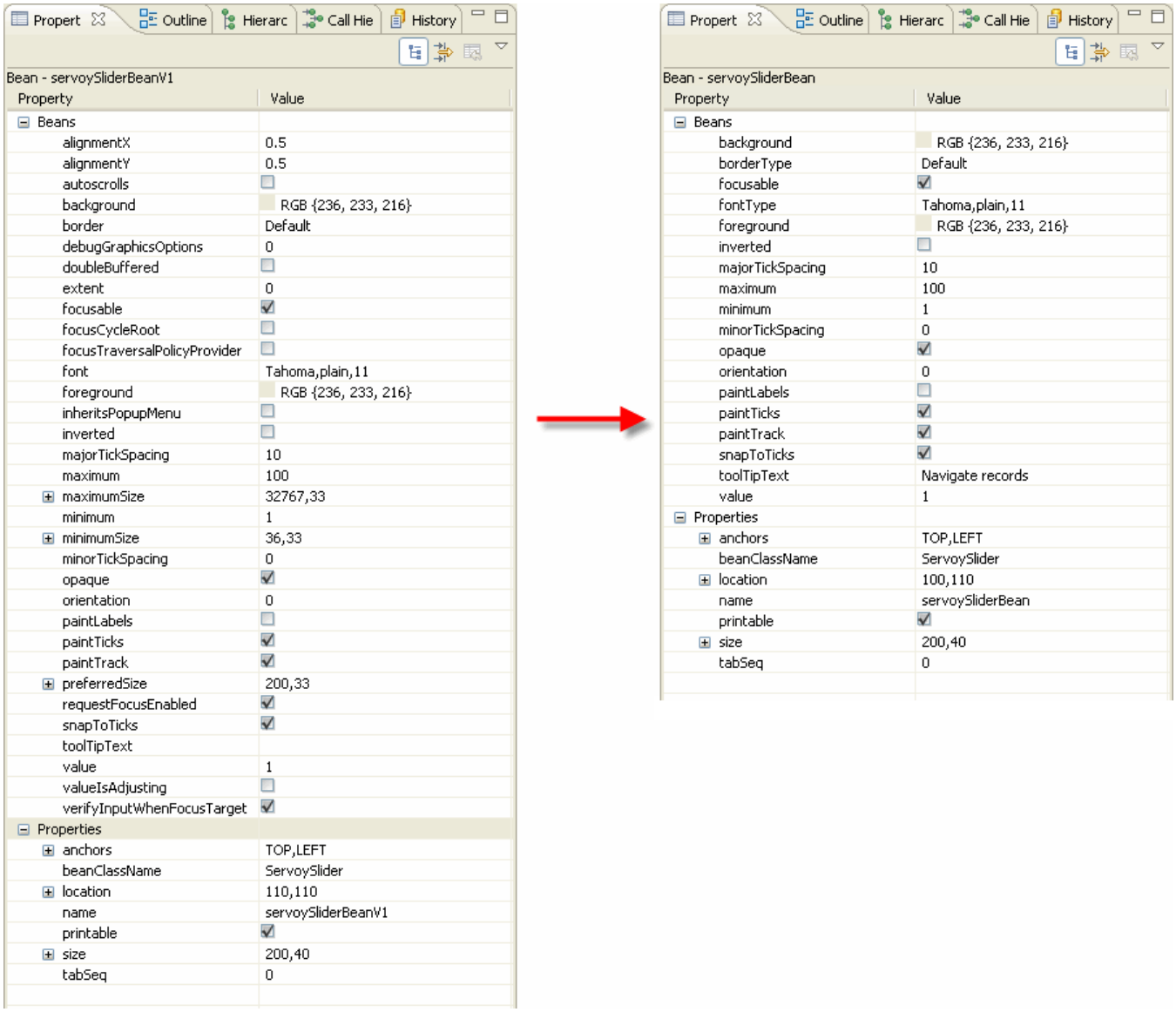
K. Polishing the bean

In the previous part of this tutorial, we created the core code of our bean to make it usable without any coding as a “navigation” bean. What we have now is a fairly usable bean which when placed on a form will automatically listen to the current foundset and follow its index as well as allow users to navigate by interacting with it.

But our conclusion last time was that our bean was still a bit rough around the edges because it showed its java implementation and lots of not really useful/usable bean properties which users will find somewhat confusing.

So our goal with this part will be to polish our bean so that it shows a more professional interface to users, just like any other of the standard Servoy components.

Basically, we want the properties of our component to change like that:



Bean - servoySliderBeanV1

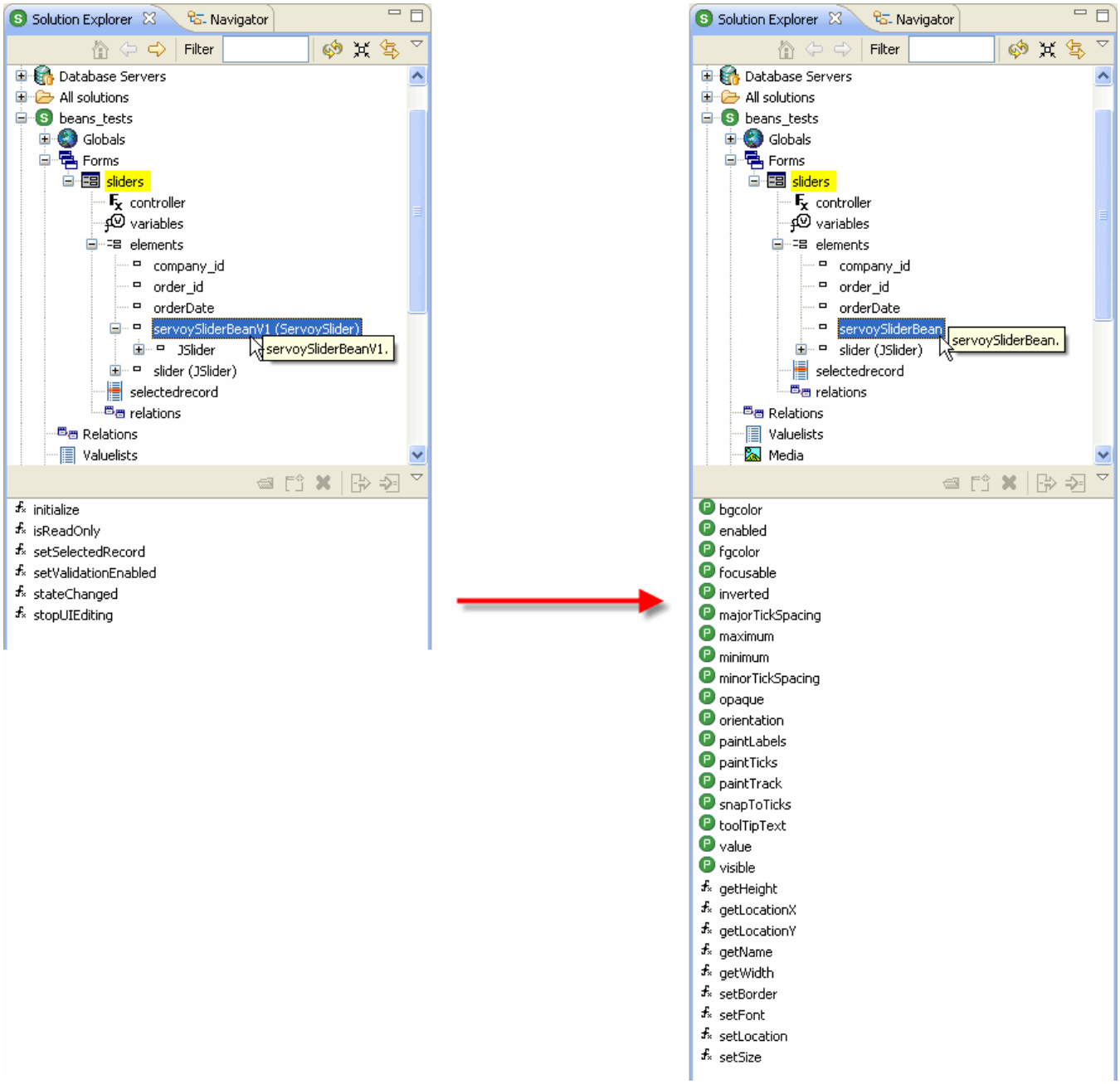
Property	Value
Beans	
alignmentX	0.5
alignmentY	0.5
autoscrolls	☐
background	RGB {236, 233, 216}
border	Default
debugGraphicsOptions	0
doubleBuffered	☐
extent	0
focusable	☑
focusCycleRoot	☐
focusTraversalPolicyProvider	☐
font	Tahoma,plain,11
foreground	RGB {236, 233, 216}
inheritsPopupMenu	☐
inverted	☐
majorTickSpacing	10
maximum	100
maximumSize	32767,33
minimum	1
minimumSize	36,33
minorTickSpacing	0
opaque	☑
orientation	0
paintLabels	☐
paintTicks	☑
paintTrack	☑
preferredSize	200,33
requestFocusEnabled	☑
snapToTicks	☑
toolTipText	
value	1
valueIsAdjusting	☐
verifyInputWhenFocusTarget	☑
Properties	
anchors	TOP,LEFT
beanClassName	ServoySlider
location	110,110
name	servoySliderBeanV1
printable	☑
size	200,40
tabSeq	0

Bean - servoySliderBean

Property	Value
Beans	
background	RGB {236, 233, 216}
borderType	Default
focusable	☑
fontType	Tahoma,plain,11
foreground	RGB {236, 233, 216}
inverted	☐
majorTickSpacing	10
maximum	100
minimum	1
minorTickSpacing	0
opaque	☑
orientation	0
paintLabels	☐
paintTicks	☑
paintTrack	☑
snapToTicks	☑
toolTipText	Navigate records
value	1
Properties	
anchors	TOP,LEFT
beanClassName	ServoySlider
location	100,110
name	servoySliderBean
printable	☑
size	200,40
tabSeq	0

With all the properties working, of course.

And we want the scripting node in the Solution Explorer to change like that:



With all the scripting and the nice built-in help and sample code working, of course.

L. Cleaning the properties

Cleaning the properties is an easy one, really, so we'll start with that.
Basically all we need to do is to provide Servoy with some Bean Info, by creating a class that will implement the java.beans.BeanInfo interface.

I'll let you have a look at the javadocs for this interface, but what you need to know is that in the absence of such a class to describe the bean, the Properties view will use what is often called "**bean introspection**" to discover the available properties and show them. You will see this process called "automatic analysis" in the javadocs of the BeanInfo interface. In any case this process relies on the Reflection API to discover the properties by analyzing all the methods with setXXX and getXXX (the getters and setters of you component) to decide whether a set of getters and getters defines a property of a bean.

But when you implement an object with the BeanInfo interface, you have the possibility to override that default mechanism and inform the properties view which one it should display (ignoring the others).

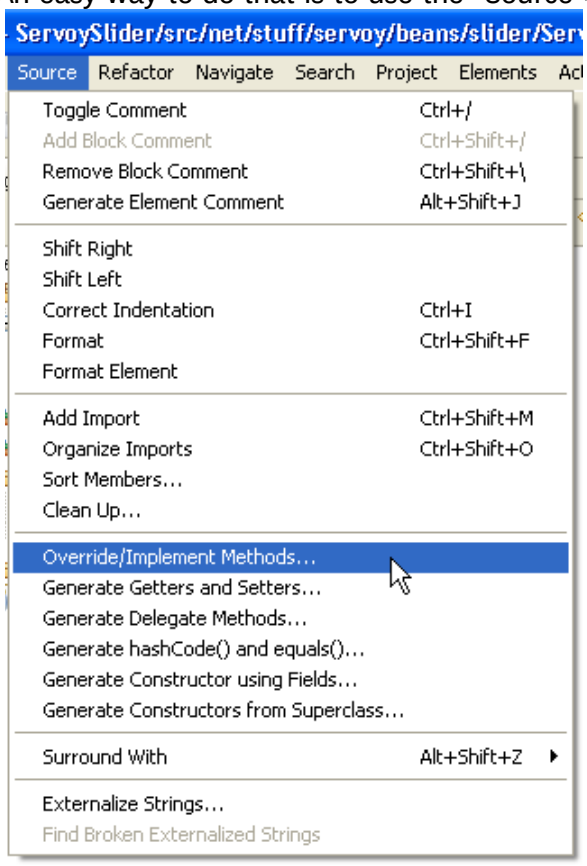
Now you could create a plain Object and implement the BeanInfo interface but you would have to code the 8 methods of its contract. Another, easier, way to do it is to subclass the java.beans.SimpleBeanInfo class which already implements this interface with default behaviour for all the methods of its contract, allowing you to override only the one method that you want, leaving the others as default. This type of class is often called an "Adapter" class, and you will find a lot of these especially when it comes to listener, when you want to override some of the event methods of an event listener interface, leaving the others as default.

So let's create our class. The only trick here is to put it in the same package as our bean, and to call it by the name of the class we want to provide properties information and add BeanInfo at the end. So in our case we need to create a class "ServoySliderBeanInfo" as a subclass of java.beans.SimpleBeanInfo, so the signature of our new class will be:

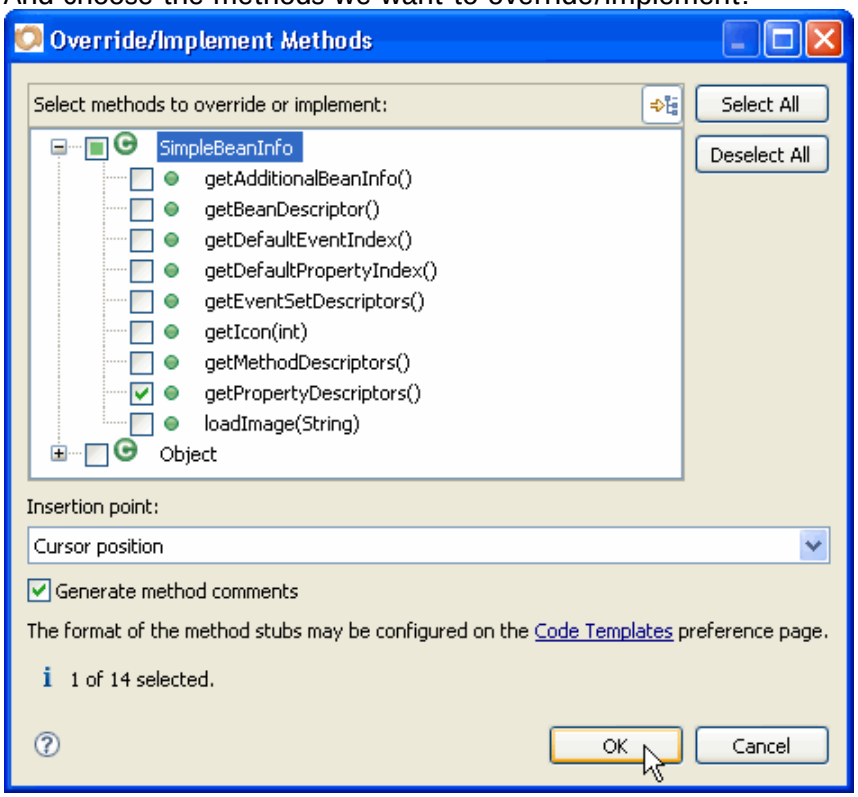
```
public class ServoySliderBeanInfo extends SimpleBeanInfo
```

I'll leave you to create this class using the "New class" dialog inside the correct package.
Now as you will see, when created this class is going to be pretty empty. So it's up to us to decide what to implement in this class.

An easy way to do that is to use the “Source > Override/Implement Methods...” functionality of Eclipse:



And choose the methods we want to override/implement:



In our case, the only method that we need to override from the SimpleBeanInfo class is the getPropertyDescriptors() method, so we check this, click OK, and Eclipse will have this stub method generated for us:

```
/* (non-Javadoc)
 * @see java.beans.SimpleBeanInfo#getPropertyDescriptors()
 */
@Override
public PropertyDescriptor[] getPropertyDescriptors() {
	// TODO Auto-generated method stub
	return super.getPropertyDescriptors();
}
```

As you see this method return an array of PropertyDescriptor objects.
You can have a look at the javadocs for java.beans.PropertyDescriptor class if you want, but in short this is the kind of object that the properties editor will need to show the relevant properties in its view.

There are few different constructors for the PropertyDescriptor object, but the one we will use take the name of the property, and the class of the “owner” of this property in parameter, so for example the PropertyDescriptor for the “background” property of our ServoySlider is going to be created like that:

```
new PropertyDescriptor("background", ServoySlider.class);
```

And since we need to return an array of all the properties, the easiest way to do that is to build it using an ArrayList, and return an array retrieved from it (you need to remember that arrays in Java are fixed size, you cannot change the size of an array once you have defined it, so that’s why using an ArrayList class, which is capable of growing as you add objects to it, is a much safer way to get an array).

So our code for all the properties we are interested in will look like that:

```
List<PropertyDescriptor> liste = new ArrayList<PropertyDescriptor>();
liste.add(new PropertyDescriptor("background", ServoySlider.class));
liste.add(new PropertyDescriptor("border", ServoySlider.class));
liste.add(new PropertyDescriptor("focusable", ServoySlider.class));
liste.add(new PropertyDescriptor("font", ServoySlider.class));
liste.add(new PropertyDescriptor("foreground", ServoySlider.class));
liste.add(new PropertyDescriptor("inverted", ServoySlider.class));
liste.add(new PropertyDescriptor("majorTickSpacing", ServoySlider.class));
liste.add(new PropertyDescriptor("maximum", ServoySlider.class));
liste.add(new PropertyDescriptor("minimum", ServoySlider.class));
liste.add(new PropertyDescriptor("minorTickSpacing", ServoySlider.class));
liste.add(new PropertyDescriptor("opaque", ServoySlider.class));
liste.add(new PropertyDescriptor("orientation", ServoySlider.class));
liste.add(new PropertyDescriptor("paintLabels", ServoySlider.class));
liste.add(new PropertyDescriptor("paintTicks", ServoySlider.class));
liste.add(new PropertyDescriptor("paintTrack", ServoySlider.class));
liste.add(new PropertyDescriptor("snapToTicks", ServoySlider.class));
liste.add(new PropertyDescriptor("toolTipText", ServoySlider.class));
liste.add(new PropertyDescriptor("value", ServoySlider.class));
liste.add(new PropertyDescriptor("location", ServoySlider.class));
liste.add(new PropertyDescriptor("name", ServoySlider.class));
liste.add(new PropertyDescriptor("size", ServoySlider.class));
```

You can see that I have used a special construct to build the List, this construct is known as “generics” and it is available in Java since version 1.5. This is fine since our bean, using the IServoyAware interface only available for 4.1.x (with java minimum requirements of 1.5) will not be usable with 3.5.x.

Basically generics force a Collection to allow only one kind of object, and will generate an Exception if you try to add another kind of object into it. This is the Java typed-language nature getting even stronger. Now this can seem counter-intuitive to JavaScript coders who get the power of JavaScript because of its loose typed variables. But the Java language gets its strength from exactly the opposite, being strongly typed allows the compiler to work more efficiently and you will know of problems at compile time with such enforcement, instead of getting errors at runtime.

For those interested in generics and the arguments for or against the use of them, you can google “java generics” and I’m sure you will find lots of interesting information about it.

For us, one of the good things about using generics is that it will allow us to retrieve an array of the correct type of objects from the List in a safe way, like this:

```
PropertyDescriptor propdescriptors[] = liste.toArray(new PropertyDescriptor[0]);
```

Note that we didn’t have to cast the result of our toArray() method.

Now to be truly on the safe side, we can wrap all our code inside a try/catch and spill the exception to the log in case we encounter it. Why? Just remember that if you do a typo in one of the name of the properties, or if these properties change later in your code, the String will not automatically be updated, it is just a String!

And how do we output the exception inside Servoy log?
Well there is a Debug class that you can use, which has some static method (you don’t need to create a new Debug() object to use it), so we will use the Debug.error(String message, Throwable ex) method in our catch() statement.

And yet another useful thing we can do with our BeanInfo class is to set alias to some properties. For example, users might not be used to deal directly with a property named “border” or “font”, in Servoy these properties in the Properties Editor View are usually named “borderType” and “fontType”, so if you want to stick with this trend, you can do it here with code like that:

```
PropertyDescriptor pd = new PropertyDescriptor("border", ServoySlider.class);
pd.setDisplayName("borderType");
liste.add(pd);
pd = new PropertyDescriptor("font", ServoySlider.class);
pd.setDisplayName("fontType");
liste.add(pd);
```

You will find the final code of our ServoySliderBeanInfo class in the downloadable Eclipse project that goes with this tutorial.

M. Adding JavaScript to our bean

Now that we have cleaned the properties of our bean, it's time to add some JavaScript access to it. But you already know how to do that, don't you?

You remember how we build a plugin, how we had a façade class implementing the IClientPlugin interface and then a real object that was the provider of the methods, implementing the IScriptObject interface.

So you know that we need to implement the IScriptObject interface and then add all the "js_" methods to our bean so that it will be scriptable just like any other Servoy component or plugin. We add the IScriptObject to the list of implemented interface for our class:

```
public class ServoySlider extends JSlider implements IServoyAwareBean,
ChangeListener, IScriptObject
```

Remember to type ctrl+space (or cmd+space on Mac) to tell Eclipse to add the import for you:

```
import com.servoy.j2db.scripting.IScriptObject;
```

And use "quick fix" to let Eclipse create the method stuff for you as well.

I will not go into the detail of implementation of the 5 methods for the IScriptObject interface contract again, just refer to the plugin tutorial for more information about it, and you will find the implementation of these in the final code of the ServoySlider class in the downloadable Eclipse project that goes with this tutorial.

The only different and interesting thing that you will find in these methods is in the getSample() method. You will see that I use a special tag "%%elementName%%" inside the Strings to refer to the final name of the object, for example:

```
if ("bgcolor".equals(methodName)) {
      buff.append("\tvar currentbg = %%elementName%%.bgcolor;\n");
      buff.append("\t%%elementName%%.bgcolor = currentbg;");
...
```

When you use this special tag, Servoy will replace it (when the user click on the getSample button of the Explorer) with the real name of your object, so that the end String that will be placed at the script editor's cursor's place will be (if our bean has a name of "servoySliderBean"):

```
      var currentbg = elements.servoySliderBean.bgcolor;
      elements.servoySliderBean.bgcolor = currentbg;
```

So you will see in the java code of the getSample() method a lot of use of this %%elementName%% tag...

So you managed to code all the 5 method of the IScriptObject interface, but as you know the real work begins now with creating all the "js_XXX" methods that will be used by Servoy's JavaScript.

There are quite a lot of them to write: almost twice as there are properties, but fortunately most of them will only have one line of code to write, a simple call to the real method.

Consider this example of the JavaScript method for the “maximum” property of our slider:

```
public void js_setMaximum(int maximum) {
	setMaximum(maximum);
}
public int js_getMaximum() {
	return getMaximum();
}
```

That’s right, nothing more is needed than just a call to the real method of the bean.
In fact most of our “js_” will only be alias to the real methods.

You must simply remember that if you want a property to appear as a property in the Solution Model, you need to create both “js_getXXX” (or "js_isXXX" for boolean) and “js_setXXX” methods.

So easily enough, if you want a property to be read only, you only need to code the “js_getXXX” method, and of course if you only want to provide a method to modify but not to read, just code the “js_setXXX” method.

Of the properties that are traditionally read only in Servoy, you will find:

- height
- locationX
- locationY
- name
- width

Of the properties that are setter methods only in Servoy, there is:

- border
- font
- location
- size

For all the others in our ServoySlider bean, we will need to add “js_getXXX” (or “js_isXXX”) and “js_setXXX” methods.

So I’ll let you code all these – you can refer to the final code in the Eclipse project anyway.

There are a few methods which need special attention though: The one that deal with Color (background and foreground), Border, and Font objects.

Why is that? Because when you use these methods in JavaScript, you don’t really use Color, Border of Font java objects, do you? In fact you are using simple Strings and pass them as parameter to the respective setters methods and expect to retrieve a String from the fgcolor and bgcolor properties as well.

What this means is that you will have to create the relevant object yourself, and return a string representation of the object from the properties of your bean.

You could build such methods yourself, and this will involve some parsing of the parameter String and creation of the relevant object, but fortunately there are two classes in Servoy to do this already (and of course there would be, otherwise how would they do it for regular Servoy elements anyway?).

The first class is the com.servoy.j2db.util.PersistHelper class, the second is the com.servoy.j2db.util.ComponentFactoryHelper class.

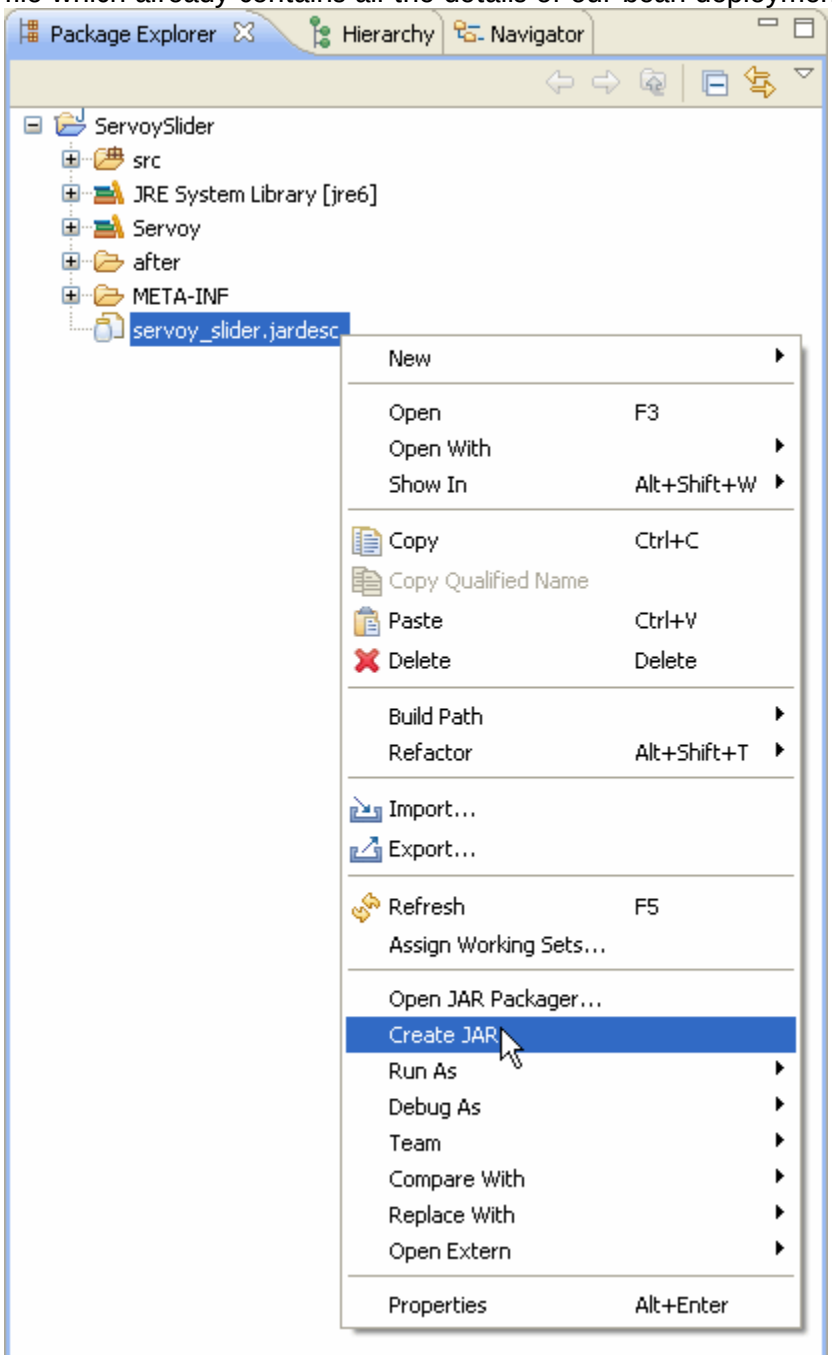
These classes are not part of the public API. I don't know why really because they are damn useful and I really think that they should be part of it. I know about it from sources of a bean that Servoy send me a few months ago as an example when I complained that there was no documentation around on how to build Plugins and Beans (long before I decided to do it myself ☺)

Anyway with the use of these two classes, you will be able to solve the problem of getting String as parameter when really what you need is an Object. So here are the methods which make use of these two classes, note the methods createColor() and createColorString() as well as the createBorder() and createFont() :

```
public void js_setBgcolor(String paramString) {
	setBackground(PersistHelper.createColor(paramString));
}
public String js_getBgcolor() {
	return (getBackground() == null) ? null :
		PersistHelper.createColorString(getBackground());
}

public void js_setFgcolor(String paramString) {
	setForeground(PersistHelper.createColor(paramString));
}
public String js_getFgcolor() {
	return (getForeground() == null) ? null :
		PersistHelper.createColorString(getForeground());
}

public void js_setBorder(String paramString) {
	setBorder(ComponentFactoryHelper.createBorder(paramString));
}

public void js_setFont(String paramString) {
	if (paramString != null) {
		setFont(PersistHelper.createFont(paramString));
	}
}
```

That's it for the JavaScript methods.

The last thing we need to do is to deploy our jar, using our previously created “servoy_slider.jardesc” file which already contains all the details of our bean deployment:



You will see that our bean now appears a lot neater in the Properties View as well as in the Solution Explorer when placed on a form.

A word of caution though: **Note** that when you modify a bean as much as what we did today, you will always need delete the one previously on the forms that used an earlier version, otherwise Servoy will try to set properties which are not there anymore (from the serialized form of the bean – open the “bean.obj” file in a text editor to see what I mean).

So before using a form with an earlier version of the bean, delete it, reinsert it from the “Place Bean...” menu or button and set the properties again. Save and everything should be ok when you launch your form.

As usual, you will find the complete Eclipse project on the Servoy Stuff web site, at this url:
http://www.servoy-stuff.net/tutorials/utils/t02/v2/ServoySlider_EclipseProject.zip
(Get rid of the previous project of the same name and import in Eclipse)

The compiled bean (targeted for java 1.5) will be available here:
http://www.servoy-stuff.net/tutorials/utils/t02/v2/servoy_slider.jar
(Put in you /beans folder)

And the little “beans_tests” solution updated to use the new bean will be available at:
http://www.servoy-stuff.net/tutorials/utils/t02/v2/beans_tests-v2.zip
(Unzip and import in Servoy 4.1.x)

Hope you enjoyed this tutorial, if not feel free to send your insults directly to me ;-)
If you have comments, suggestions, special requests send me a note.

The next part of this tutorial will be about preparing the ground for usage in the Web client.
So get ready for that and in the meantime have good fun polishing your beans ☺!

Patrick Talbot
Servoy Stuff
2009-07-04