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

PART 8

NN. The last part!

In the previous parts of this tutorial we build a Servoy Aware bean based on a Swing JSlider for the Smart client and based on a Wicket Panel and TextField + a dynamic JavaScript library to emulate the JSlider in the Web client. We now have a pretty usable bean, which responds quite well in both environments and we saw lots of interesting stuff along the way.

But there was one thing we didn't touch and someone made a remark about it on the Java for Servoy Developers Blog. So I decided to add yet another part to this tutorial to give our bean the finishing touch: event handling, or how to attach JavaScript Servoy methods to internal events of our bean to make it even more open and user friendly.

The idea of call-backs is that instead of calling the bean repeatedly to test if some of its value has changed (or some other internal events), we ask him to call us back, or more precisely to call one of our JavaScript method back when some event occur.

This is the "don't call me, I call you" principle well known in Object Oriented patterns - and by every girl I tried to date when I was young, unfortunately :{

But before we get to the meat of it, there is something I want to change in the Servoy Slider as it is of now: our methods to access the dataProvider were trying first to test if the dataProvider was part of the parent foundset; which, in case of globals and related dataProviders was simply wrong, as I have learned recently by some exchange with Johan Compagner (see this post on my blog to learn more about it: http://www.servoy-stuff.net/blog/index.php?entry=entry090811-054744)

Don't worry! It's going to an easy change because as you might remember, we put our test method in one place, our utility class: the ServoyUtils class...

If you recall, the method was:

```
public static boolean isContainedInFoundset(IRecord record, String dataProviderID)
{
	if (record != null && isNotEmpty(dataProviderID)) {
		IFoundSet foundset = record.getParentFoundSet();
		if (isExisting(foundset, record)) {
			boolean isContained = false;
			String[] providers = foundset.getDataProviderNames(
				IFoundSet.COLUMNS);

			if (providers != null) {
				for (int i = 0; i < providers.length; i++) {
					if (dataProviderID.equals(providers[i])) {
						isContained = true;
						break;
					}
				}
			}
			if (isContained && foundset.getSelectedIndex() > -1) {
				return true;
			}
		}
	}
	return false;
}
```

Here we tested if the record was not null and the dataProvider not empty, then if the foundset was not null and contained records, and then we retrieved a String[] array of all the dataProvider's name (of type COLUMNS) for this foundset and looped to test for equality with our dataProvider String.
The fact is that it doesn't take into account globals, related fields, calc and aggregates, when the IRecord interface's getValue() (and setValue()) methods already know how to deal with these!
So we are going to change our method like this instead:

```
public static boolean isContainedInFoundset(IRecord record, String dataProviderID)
{
	boolean isContained = false;
	if (record != null && isNotEmpty(dataProviderID)) {
		try {
			if (isGlobal(dataProviderID)) {
				isContained =
record.getParentFoundSet().containsDataProvider(dataProviderID);
			} else {
				if (dataProviderID.indexOf(".") < 0) {
					isContained = ((Scriptable)record).has(
						dataProviderID, ((Scriptable)record));
				} else {
	// no way to know for sure if the dataProvider is valid in related records.
	// so let's consider it is - it will not harm since calls to
	// getValue() or setValue() on an IRecord will not throw Exceptions!
					isContained = true;
				}
			}
			} catch (Exception ex) {
				Debug.error(ex);
			}
		}
		return isContained;
	}
```

The isGlobal() utility method just test for the magic prefix “globals.” for a String:

```
public static boolean isGlobal(String s) {
	return (s != null && s.trim().toLowerCase().startsWith("globals."));
}
```

This time we test if record is not empty (we need one to ask him to test the dataProvider) and if the dataProvider is not empty of course, then if it is a global we ask the parent IFoundSet containsDataProvider() method (which is inherited from IGlobalValueEntry) to test for its existence.
If it is not a global, then if there is a dot inside the String it is a related record, otherwise we can cast the record to a Scriptable object (a Rhino object) to see if it contains the property.
If the value contains a dot, then there is nothing we can do to be sure that the dataProvider exists, since the related IFoundsets might just be null... So we won’t be able to ask them to test for the value...

It is a bit complicated, I know, and I would much prefer either the getValue() of IRecord to throw an Exception if the dataProvider doesn’t exist or to have a reliable containsDataProvider() method (either on the IRecord or on the parent IFoundSet) returning an accurate response for all cases...
But it doesn’t seem to be possible since Servoy internally doesn’t care if a dataProvider exist or not (it never throws Exceptions).

That’s it for the changes to our existing code; now let’s look at event handling...

OO. Preparing for event handling

When looking at events related to a bean we need to ask ourselves: what kind of events is really relevant in our case? Is the onFocusGained or onFocusLost, or onAction events really making sense for our Slider and are we going to be able to implement them reliably in the Swing bean as well as the Wicket bean? I don’t think this is worth the effort but if you do, please implement them, from the sources provided you could give it a shot.

For now we are going to focus on the onDataChange event, and see how we can do it.

First, to be able to call back a Servoy method, we need a property in the bean to store its value.
Now the value could be a String (name of the method, possibly starting with “globals.” for global methods, or it could be something else (a Scriptable object = a Rhino Function) if only we could use the Servoy method property editor.
It is supposed to be implemented in Tano, but it’s still not here yet in the beta, I have asked for it again, and I keep my fingers crossed to see it in the final, but in the meantime we will use a simple String property to hold the method’s name, so let’s add it in our ServoySlider class:

```
private String onDataChange;
```

Then with the help of Servoy’s Eclipse Source menu, let’s generate the getter/setter:

```
public String getOnDataChange() {
	return this.onDataChange;
}

public void setOnDataChange(String onDataChange) {
	this.onDataChange = onDataChange;
}
```

We will change the setter in a minute.

First let's add these 2 method signatures to our IServoySliderBean interface:

```
public String getOnDataChange();
public void setOnDataChange(String onDataChange);
```

As soon as you do that the ServoySwingSlider and the ServoyWidketSlider classes will be displayed with errors in the Package Explorer.
No worries, let's add the same property in both class and generates the getter/setter too!

Once we're done, we can alter our ServoySlider setter to this:

```
public void setOnDataChange(String onDataChange) {
	this.onDataChange = onDataChange;
	if (component != null) {
		component.setOnDataChange(onDataChange);
	}
}
```

And add a call to it at the end of our initComponent() method:

```
	setOnDataChange(onDataChange);
}
```

While we're here, don't forget to add the declaration of our new property in the ServoySliderBeanInfo getPropertyDescriptors() as well:

```
liste.add(new PropertyDescriptor("onDataChange", ServoySlider.class));
```

PP. Preparing for event handling

Now that we've got a property to hold the name of the Servoy method to call back, we need to ask ourselves when we need to fire it. We set ourselves to a dataChange event, so it needs to be fired every time the value is changing.

In the Swing class, where are we going to put that trigger? Well, we already have an internal event handling since our bean is a ChangeListener to itself, remember? So it seems like the ideal place to put our call to the Servoy method.

What we need to ask ourselves now is: are we going to fire the event each time a ChangeEvent event is received or only when the valueAdjusting is false (meaning when the user has finished playing with the handle of the Slider)? I would vote for the latter but you can do it differently if you prefer.

Also one thing we need to take into account is that for an onDataChange event, Servoy's users are expecting to receive 2 values in parameters: the "old value" and the "new value". But right now we are not holding the previous value, so we need to do it, and add another property to our ServoySwingSlider class:

```
private Number previousValue = 0;
```

I set it to be a Number because then it can be a Double as well as an Integer: perfect for what we want.

Of course we need to set our previous value each time we receive it from a setSelectedRecord() call from Servoy, so let's add it to our setSliderValue() method:

```
	private void setSliderValue(Number x) {
		ignoreUpdate = true;
		previousValue = x;
		if (isUsingFactor()) {
			setDoubleValue(x.doubleValue());
		} else {
			setValue(x.intValue());
		}
	}
```

Now in our stateChanged() method we can use this previousValue to test if it is not equal to the new value (in which case there is no need to fire a dataChange event, is there?) Meaning we need a safe method to safely compare numeric values, so we first add it to our ServoyUtils class:

```
	public static boolean numberEquals(Number a, Number b) {
		if (a == null && b == null) return true;
		if (a == null || b *= null) return false;
		return (a.doubleValue() == b.doubleValue());
	}
```

We compare on double values for all cases.

Now we can use our method in the stateChanged method (changes in red):

```
public void stateChanged(ChangeEvent e) {
	if (getValueIsAdjusting()) {
		// code not changed here
	} else {
		if (!ignoreUpdate
				&& ServoyUtils.isExisting(currentFoundset, currentRecord)) {
			Number newValue = null;
			Number previous = previousValue;
			if (ServoyUtils.isContainedInFoundset(
					currentRecord, getDataProviderID())) {
				if (currentRecord.startEditing()) {
					if (isUsingFactor()) {
			newValue = getNumberValue();
			currentRecord.setValue(getDataProviderID(), newValue);
					} else {
			newValue = getValue();
			currentRecord.setValue(getDataProviderID(), newValue);
					}
				}
			} else {
				previous = currentFoundset.getSelectedIndex() + 1;
				newValue = getValue();
				currentFoundset.setSelectedIndex(getValue()-1);
			}
			if (ServoyUtils.isNotEmpty(onDataChange)
				&& !ServoyUtils.numberEquals(previous, newValue)) {
				ServoyUtils.runFunctionByName(onDataChange, app,
					new Object[] { previous, newValue});
			}
			previousValue = newValue;
			if (app != null) {
				app.setStatusText("Done");
			}
		}
	}
```

```
	ignoreUpdate = false;
}
```

I declare a newValue and previous Number variables, set them to the appropriate value depending on the context, and then I test if an onDataChange String has been provided, and if the previousValue is different from the newValue. If so, I call another utility method, that we will add the ServoyUtils utility class (because we will need the same code in the Wicket bean of course). Then in the end I set the previousValue to the new Value for the next time...

This method takes the name of the Servoy method (our onDataChange String, the IClientPluginAccess object – we stored a reference to it in the app variable, and an array of Object[] parameters which will be send to the Servoy method.

QQ. Callback to Servoy

Now let's look at the implementation of this runFunctionByName() method that we add to our ServoyUtils class:

```
public static void runFunctionByName(String functionName,
		IClientPluginAccess app, Object[] params) {

	if (functionName != null) {
		String fName = functionName.trim();
		if (fName.length() > 0) {
			if (isGlobal(fName)) {
				runGlobalFunctionByName(fName, app, params);
			} else {
				runFormFunctionByName(fName, app, params);
			}
		}
	}

}
```

The public method will forward to 2 specialized methods because the call to a form method is different from the call to a global method, so here we test that we got some valid objects, and we check if the method is a global (starting with the magic "globals." CharSequence) or a form method.

Let's see the runGlobalFunctionByName first:

```
private static void runGlobalFunctionByName(String fName,
		IClientPluginAccess app, Object[] params) {

	try {
		fName = fName.substring("globals.".length());
		app.executeMethod(null, fName, params, false);
	} catch (Exception ex) {
		Debug.error(ex);
	}

}
```

Inside a try/catch block we strip the name of the method of the “globals.” magic CharSequence, then we simply make a call to the executeMethod of Servoy’s IClientPluginAccess interface.
The first argument is the name of the form (not relevant here, since we call a global method), the second is the function name, and then we can add an Object[] array of params, this will be the old and new values (that will be retrieved in the Servoy method as arguments[0] = previousValue and arguments[1] = newValue), and we can set the call to be asynchronous if we need to.

The runFormFunctionByName is quite similar:

```
private static void runFormFunctionByName(String fName,
            IClientPluginAccess app, Object[] params) {

      IFormManager manager = app.getFormManager();
      if (manager != null) {
            IForm form = null;
            int indexOfDot = fName.indexOf('.');
            if (indexOfDot > -1) {
                  form = manager.getForm(fName.substring(0, indexOfDot));
                  fName = fName.substring(indexOfDot+1);
            } else {
                  form = manager.getCurrentForm();
            }
            if (form != null) {
                  String formName = form.getName();
                  if (formName != null) {
                        try {
                              app.executeMethod(formName, fName, params, false);
                        } catch (Exception ex) {
                              Debug.error(ex);
                        }
                  }
            }
      }
}
```

This time we need to use the IFormManager to get a form and check/retrieve its name, we got one by the getFormManager() method of the IClientPluginAccess interface.

Then we check if our function name contains a dot. If so, this means that the function might be inside a form that is not the current one (actually it can be useful when a form is used in a tabPanel). If so, we retrieve a form by its name (the part of the function name before the dot), using the getForm(name) method of the IFormManager.
If the method is in the current form, we can retrieve it using the manager.getCurrentForm() (similar to currentController of the globals node of the Servoy Solution explorer).

One we have a form we can easily retrieve its name, and then call the executeMethod of the IClientPluginAccess interface, this time with the formName as first parameter instead of null.
Not that the fName variable, if expressed with something like nameOfForm.myCallBackMethod, will be stripped to myCallBackMethod, while nameOfForm will hopefully has been retrieved by the IFormManager getForm(name) method.

So that’s it for our method call back handling. And that’s it for our ServoySwingSlider implementation of an onDataChange method. Last thing to do is to implement the same thing in the ServoyWicketSlider as well.

RR. DataChange event in the Wicket bean

In the ServoyWicketSlider things are going to be even easier, because we added the onDataChange String property already, and we already have a previousValue that we used already to test if the value had changed, and of course we have all the utility methods in our ServoyUtils class...

We only need to keep a reference to the app that will be send to us in the initialize() method, so we add a property to hold it:

```
	private IClientPluginAccess app;
```

Then we set it in our initialize() method:

```
	public void initialize(IClientPluginAccess app) {
		this.app = app;
	}
```

And now we can add our call back code to Servoy in the setNumberValue() method (changes in red):

```
public void setNumberValue(double numberValue) {
	this.numberValue = numberValue;
	if (previousValue != numberValue) {
		if (ServoyUtils.isExisting(currentFoundset, currentRecord)) {
			if (ServoyUtils.isContainedInFoundset(currentRecord,
			getDataProviderID())) {
				Number newValue = null;
				Number previous = previousValue;
				if (currentRecord.startEditing()) {
					if (isUsingFactor()) {
						newValue = numberValue;
			currentRecord.setValue(getDataProviderID(), newValue);
					} else {
						newValue = getCurrentValue();
			currentRecord.setValue(getDataProviderID(), newValue);
					}
					setChanged();
				}
			} else {
				previous = currentFoundset.getSelectedIndex() + 1;
				newValue = getCurrentValue();
				currentFoundset.setSelectedIndex(getCurrentValue()-1);
			}
			if (ServoyUtils.isNotEmpty(onDataChange)) {
				ServoyUtils.runFunctionByName(onDataChange, app,
					new Object[] { previous, newValue } );
			}
			previousValue = Utils.getAsDouble(newValue);
		}
	}
}
```

That's it for the Wicket component. Now all you have to do is deploy in Servoy, put the bean on a form, set a few parameters and add an onDataChange parameter with the name of a function...

As usual, you will find the complete Eclipse project on the Servoy Stuff web site, here:
http://www.servoy-stuff.net/tutorials/utils/t02/v6/ServoySlider_EclipseProject-v6.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/v6/servoy_slider.jar
(Put in you /beans folder)

And the little “beans_tests” solution updated to use the new bean in situation will be available at:
http://www.servoy-stuff.net/tutorials/utils/t02/v6/beans_tests-v6.zip
(Unzip and import in Servoy 4.1.x)

I really hope you enjoyed this tutorial serie, I spend a lot of time doing it and I hope that you will have find it useful and that you will have learned a thing or two along the way!

SS. What’s next?

Well, with the Servoy Tano 5.0beta1 already in the testing loop and the big changes in the API, I would say that this is not the end of the story for plugins and beans! This is certainly not the end of the possible Java enhancements that I intend to bring to the platform… Because if Tano is great and adds a lot of very nice features, I still consider it as a RAD java tool and as such there will always be something missing, see the last SVG Bean that I did on request, and the request that I get from users now and then for a special plugin or bean that would do what cannot be in the standard distribution.

That’s the idea behind every great framework (and in a sense that’s what Servoy is, if you consider the public API): put in the core all the heavy functionalities that have the consensus of everyone, and give extensions possibilities in the form of a rich API, and you will see peripheral projects spawning and adding all the little niceties that make a platform a real solution for any kind of applications… Look at Wicket for example: there is the Wicket core and then there are the numerous projects around it, like wicket-extensions, wicket-stuff ;-) and wicket-minis. I think that Servoy will gain from Open Source third party projects (in fact it already has!).

I dream of real Web compatibility, I envision using Servoy as the back-end for other technologies, like Flex for example (also Java based, and oh so sexy!) and I foresee better, more intelligent, more focused components, I will attempt to do some, maybe you will join me in this Open Source effort too.

And there is the documentation, still the weakest point of the platform, that will need some common efforts, maybe the upcoming Wiki (hopefully ready and enhanced for 5.0 will be ready for the final Tano release) will be open enough that we will all be able to add our tips and tricks and comments and links to build a centralized source of information and documentation.

So keep visiting the Servoy Stuff website, send me your comments and questions and requests, I really enjoy hearing from you!

Patrick Talbot
Servoy Stuff
2009-08-18