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

PART 1

A. Introduction

This tutorial is going to look into details on how to build beans for Servoy.
We are not going to see in particular how to create a Swing bean (there are lots of available books and resources which will give you this information - you can use any kind of beans because as you will see you can use “normal” java beans coming from any source, being standard Swing beans, or third parties open source or commercial components), no, we are going to build what we will call a “Servoy-aware” bean, meaning a bean adapted for the Servoy platform to facilitate its use in Servoy and allow it to communicate back with Servoy clients.

This first part is going to look at how to use regular beans in the Servoy IDE, how to script them and what you can do with them, which is quite a lot already. But we will see some of the shortcomings of using beans not adapted for Servoy and the amount of code necessary each time you want one of these “regular” beans to be integrated into the Servoy environment.
In the next parts we will take one bean in particular and start to extend it to make it “Servoy-aware”, we will also see how we can add scripting, call-backs, and so on, and then we will try to make it fully web compatible by creating a wicket version of the bean, usable in the web client so that our final bean will be usable on any Servoy clients.

The bean will be Open Source of course and you will find along with this tutorial everything you need to build it yourself: sources, resources, eclipse project to use in the Servoy IDE, and a sample solution to demonstrate its use.

That’s a big agenda here, and although I have a rough plan in my head right now, I still have no idea what we are going to end up with, really. But I’m optimistic, so trust me, whatever it is, we will see along the road a lot of useful techniques and explanations that will help you build your own.

So let’s start simple by getting accustomed to “regular” beans inside the Servoy platform.

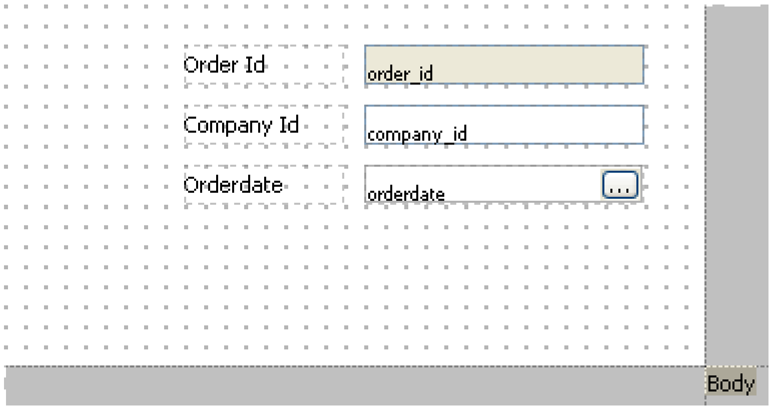
B. The Sample / Test project

First, in Servoy 4.1.x let’s create a new Servoy solution; we will name it “beans_tests”.
The name reflects well the fact that this is the solution that we will use to test our “regular” beans as well as our own “Servoy-aware” bean, it will allow us to switch from “regular” to “Servoy-aware” beans and compare their behaviours and see the result of our coding as we go along, and in the end it will allow us to test our bean in a real context.

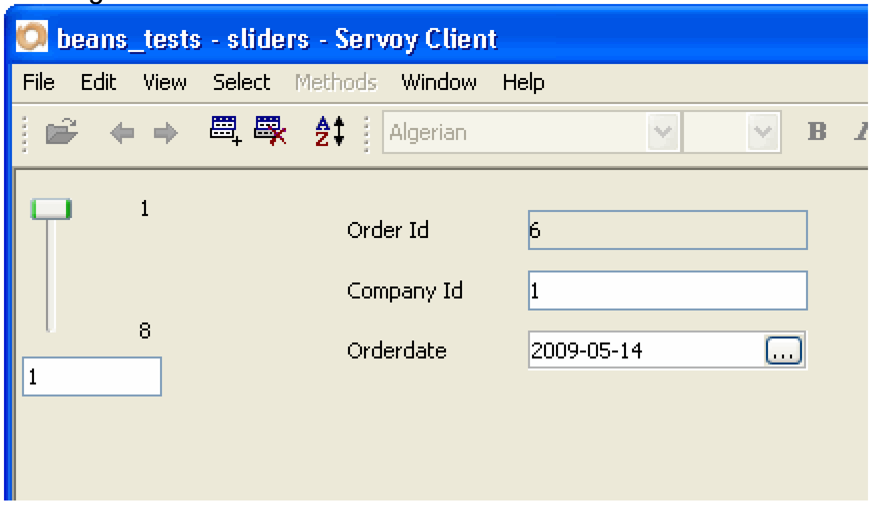
Inside our “beans_tests” let’s add a form, and call it “sliders”, since we are going to put test some sliders on it. Let’s use the “udm” database “orders” table, since it’s a standard sample db that you should already have with your Servoy installation. Of course, you can base it in any other database and table of course but in that case you will probably have to tweak some of the methods later.

Let's choose a few fields to place on our form so that it is not too empty, I have chosen "order_id", "company_id" and "orderdate" (I think there is a reason why these fields seems to be duplicated, if you really need to know I suppose you will have to search in the CRM sample solution).

So we now have this kind of form:

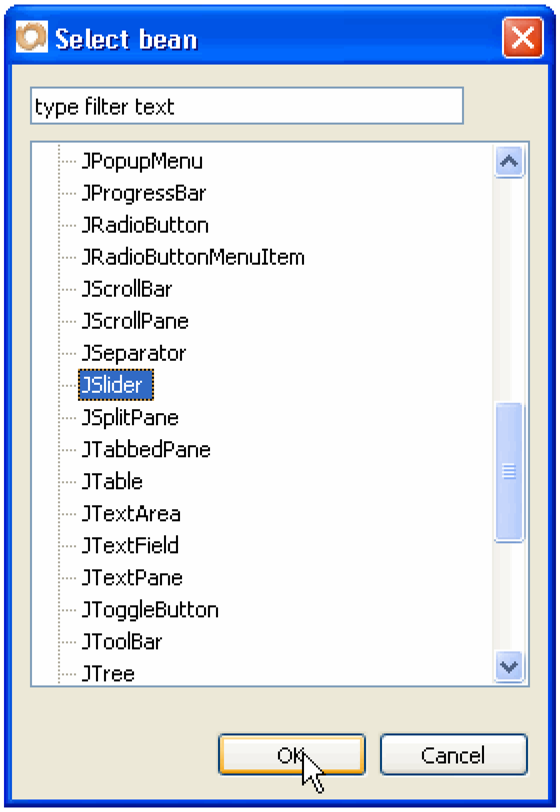


Looking like this in the smart client:


beans_tests - sliders - Servoy Client
File Edit View Select Methods Window Help
Algerian
B
1
8
1
Order Id
6
Company Id
1
Orderdate
2009-05-14
...

Not very exciting, is it?

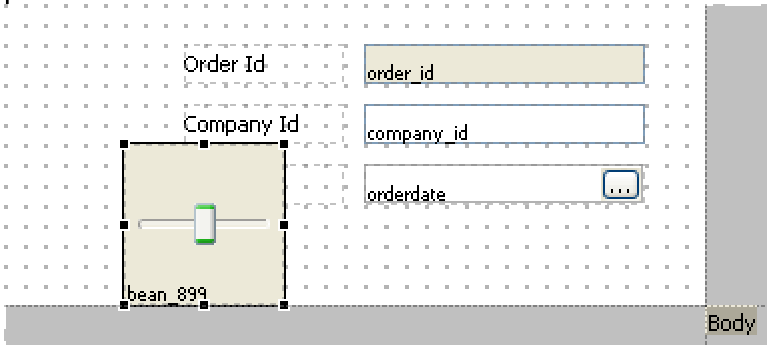
We are going to add a slider bean to use as navigation, just like the Servoy default navigator does. So click on the "Place bean..." button or choose "Place Bean..." from the elements menu, then in the list of available beans, choose the "JSlider":



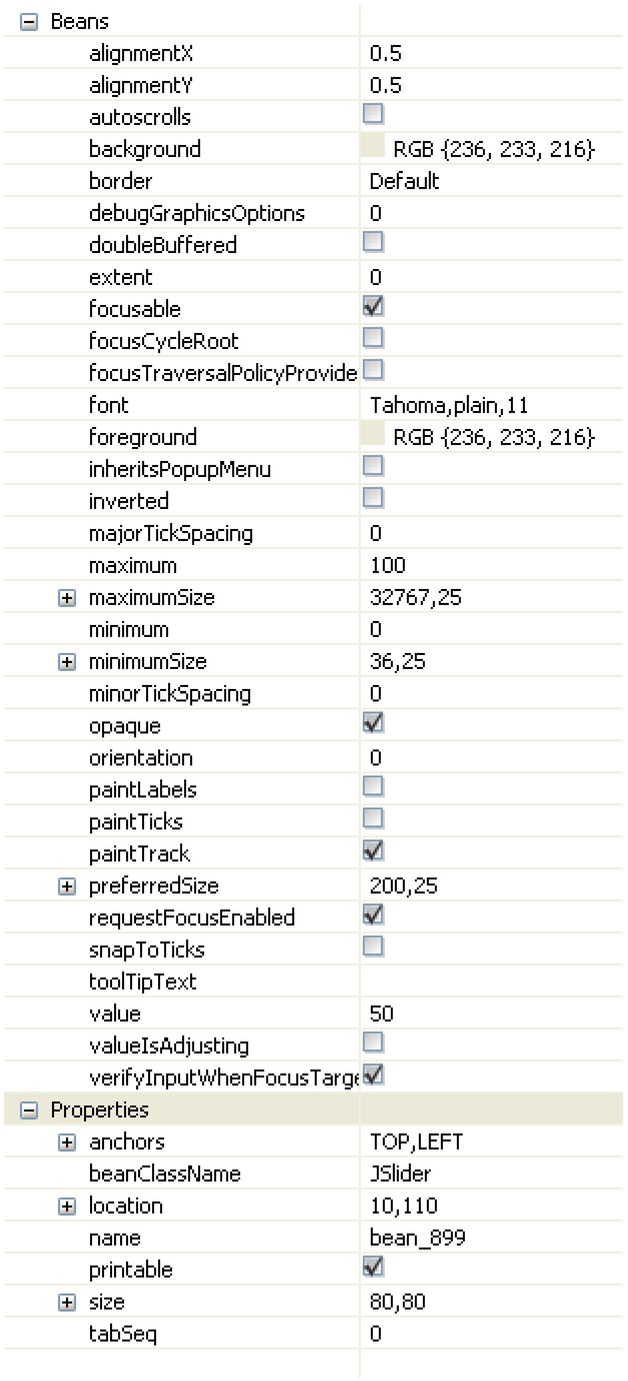
You can see that there are a lot of these JSomething beans in there. These are the standard swing beans, if you take a look at your /beans folder, you will find a “swingbeans.jar”, it’s size is only 2kb and if you open it, you will see that there is only a MANIFEST.MF which only declares the javax.swing related component as java beans.
There is no code. This is because the javax.swing components are already part of you java installation.

Now none of these swing beans are Servoy-aware, but they are certainly usable as such, providing you script them as we will now with our JSlider.

The JSlider has been placed on our form with a name of “bean_” + an arbitrary number at an arbitrary place with a size of 80x80:



The size of 80x80 is always the same whatever the bean you are selecting. And this is actually a bug of Servoy (I submitted a case #**211901** a few weeks ago about this, never head of it since): ignoring the "preferredSize" hints provided by every bean, in that case it should be 200x25 as you can see form the properties:



⊟ Beans	
alignmentX	0.5
alignmentY	0.5
autoscrolls	☐
background	RGB {236, 233, 216}
border	Default
debugGraphicsOptions	0
doubleBuffered	☐
extent	0
focusable	☑
focusCycleRoot	☐
focusTraversalPolicyProvide	☐
font	Tahoma,plain,11
foreground	RGB {236, 233, 216}
inheritsPopupMenu	☐
inverted	☐
majorTickSpacing	0
maximum	100
⊞ maximumSize	32767,25
minimum	0
⊞ minimumSize	36,25
minorTickSpacing	0
opaque	☑
orientation	0
paintLabels	☐
paintTicks	☐
paintTrack	☑
⊞ preferredSize	200,25
requestFocusEnabled	☑
snapToTicks	☐
toolTipText	
value	50
valueIsAdjusting	☐
verifyInputWhenFocusTarge	☑
⊟ Properties	
⊞ anchors	TOP,LEFT
beanClassName	JSlider
⊞ location	10,110
name	bean_899
printable	☑
⊞ size	80,80
tabSeq	0

C. The JSlider properties

The "Properties" part at the bottom contains regular Servoy properties. The "Beans" part contains the specific properties exposed by the bean, so let's review them:

- **aligmentX / alignmentY**: this is the standard way of setting the alignment using a float value between 0 and 1, the meaning is 0 = left/top – 0.5 = center/middle – 1 = right/bottom – you can use any values in between to better tweak the alignment
- **autoscrolls**: this boolean is just a hint, there is no such thing as an automatic scrollable component in Swing, if you need a component to scroll (showing scrollbars) you will have to embed it into a JScrollPane
- **background**: the color to use as background if the bean is set to “opaque”
- **border**: can be any of the standard borders used by Servoy (Etched, Line, Bevel, etc.)
- **debugGraphicsOptions**: an integer that can be used when debugging graphic operation in java when creating a component (values comes from JComponent : BUFFERED_OPTION, FLASH_OPTION, LOG_OPTION, NONE_OPTION) default to 0/NONE_OPTION – no use in Servoy anyway – see java Swing API for more info
- **doubleBuffered**: boolean to set whether the bean should use a buffer to paint. The technique known as doubleBuffering is especially useful when doing animations, because you don’t paint directly on the surface of the screen but into an offline buffer, and then quickly copy the result into the paint surface. You will not see any notable difference for a bean since it also depends on the container (frame window) to use a buffer or not, if the window does use a buffer then the components will automatically use it as well.
- **extent**: this is the total amount of values the slider represents; leaving it to 0 will ask the slider to deduce it from the values of maximum – minimum.
- **focusable**: boolean, if true, the component will be part of the focus sequence, if not you will not be able to access it using the focus keys (tab / shit-tab)
- **focusCycleRoot / focusTraversalPolicyProvider**: used if a component is a composite of different component and is a focus root and provides a policy to traverse its inner components (no use in the case of a JSlider) – see the swing API to learn more about the Focus system
- **font**: set the font of the labels if any
- **foreground**: the color of the labels if any
- **inheritsPopupMenu**: if the parent container has set a PopupMenu, this boolean will tell if the Slider will also react to it
- **inverted**: will invert the direction of the slider know; this will be used when orientation is vertical to allow increasing the value when moving down from top to bottom, instead of the standard behaviour which is up from bottom to top.
- **majorTickSpacing**: allow to add big tick marks for every x values
- **maximum**: the maximum value represented by the slider
- **maximumSize**: the maximum size of the bean on a form
- **minimum**: the minimum value represented by the slider
- **minimumSize**: the minimum size of the bean on a form
- **minorTickSpacing**: allow to add little tick marks for every x values
- **opaque**: set the background to be painted with the background value or be transparent
- **orientation**: 0 is horizontal – 1 is vertical
- **paintLabels**: will add a label with the related value for every major ticks
- **paintTicks**: will paint the major and minor ticks if the related spacing is > 0
- **paintTrack**: will paint the line of the slider or not
- **preferredSize**: a hint for bean IDE to set the initial value (this is the optimal size of the bean when it has all its default values) see above about Servoy not using this hint
- **requestFocusEnabled**: a hint to the container Focus System, that this bean will respond to focus event – see the javadocs for JComponent for more info on this
- **snapToTicks**: boolean, if true, will make the slider jump to the closer tick if set to an in-between value
- **value**: int, the current value represented by the slider

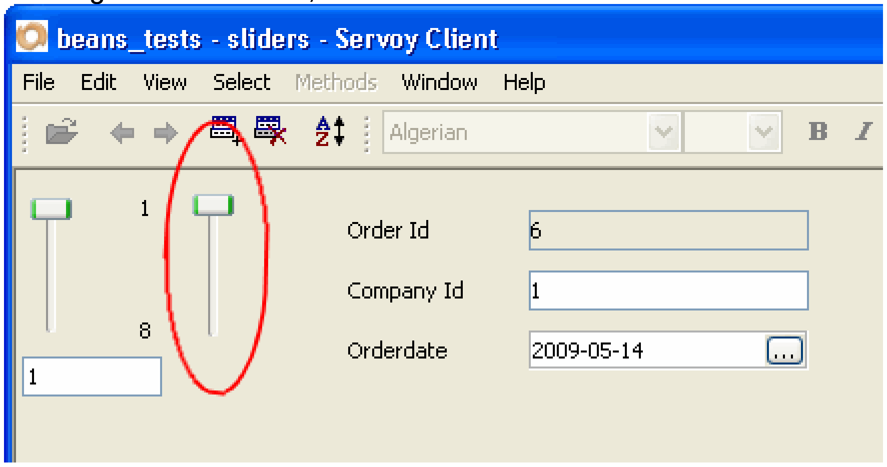
- **valueIsAdjusting**: boolean indicating that the slider knob is currently dragged, so the value is not to be taken into account, not to be used in the properties since it is a "dynamic" which will be set by the component
- **verifyInputWhenFocusTarget**: can be used internally by components to adjust validation, not used in the case of a JSlider – see JComponent javadocs for more info on this

Now that we know what all these properties mean, let's tweak our slider.
For example, if we want it to be just like the regular Servoy default Navigator, we will adjust its values like that:

- maximum = 8
- maximum = 1
- inverted = true
- orientation = 1
- value = 1
- size = 25,80
- name = slider

Note that in the udm "orders" table that we based our form on, there is 8 records, so that's why I set the maximum to 8, you might want to set it to another value if your form is based on another table or if this table contains another number of records

Starting to look familiar, isn't it?

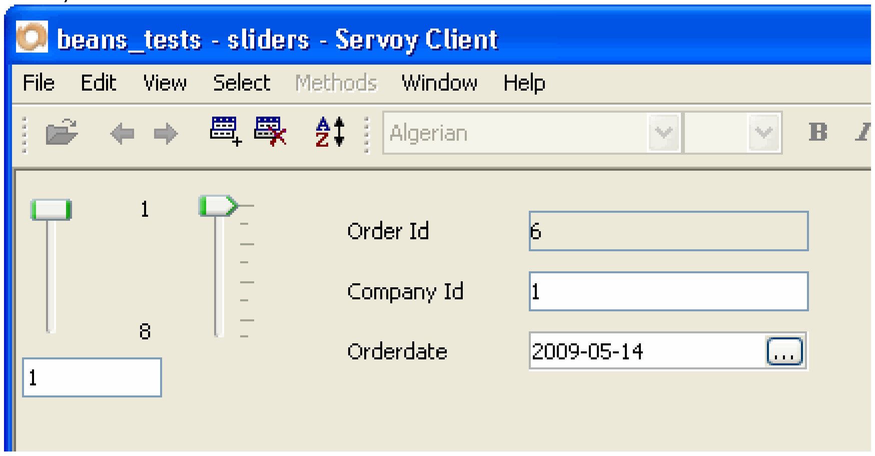

beans_tests - sliders - Servoy Client
File Edit View Select Methods Window Help
Algerian
B I
1
8
1

Order Id: 6
Company Id: 1
Orderdate: 2009-05-14

Now let's add a few things to make it look even better:

- majorTickSpacing = 2
- minorTickSpacing = 1
- paintTicks = true
- snapToTicks = true
- size = 40,80

Now, it should look like that:


beans_tests - sliders - Servoy Client
File Edit View Select Methods Window Help
Algerian
1
8
1
Order Id
6
Company Id
1
Orderdate
2009-05-14

As you can see, depending on the fact that ticks are visible or not, the shape of the slider's knob changes to get a pointy head. You can also see the effects of setting major and minor ticks, as well as the effect of the snapToTicks boolean when you move the slider with the mouse: it will refuse to stay in-between two ticks, and will jump to the closest mark (with a slight preference to the "next" value, - this is the effect of rounding a float/double value to an int in the inner implementation of the slider).

D.Scripting the JSlider in Servoy

This is all well and good, but right now our slider is not very useful, because whatever value you set it too, it does not drive the foundset, nor take its value from the selected record.

To achieve that, we will need to add some scripting to our form. First let's attach it to the size of our foundset, this way we don't need to guess in advance what values to put for maximum and tick spacing.

Add a form method to the "onShow" event; let's call it FORM_onShow, with this code:

```
function FORM_onShow()
{
	elements.slider.minimum = 1;
	var max = foundset.getSize();
	elements.slider.maximum = max;
	// we want a maximum of 10 major ticks:
	var tickSpacing = Math.round(max/10);
	elements.slider.majorTickSpacing = tickSpacing;
	// and no minor ticks:
	elements.slider.minorTickSpacing = 0;
}
```

As you can see, you can set/get the values of any properties of a bean exactly the same way you would do with a regular Servoy field element.

Now let's hook the value of the bean to the selected record.

Add a form method to the "onRecordSelection" event, let's call it REC_selectRecord, with this code:

```
function REC_selectRecord()
{
	elements.slider.setValue(controller.getSelectedIndex());
}
```

Now when we navigate our form with the standard “navigator” or use one of the standard form menu (Previous Record or Next Record), our slider’s knob is following.

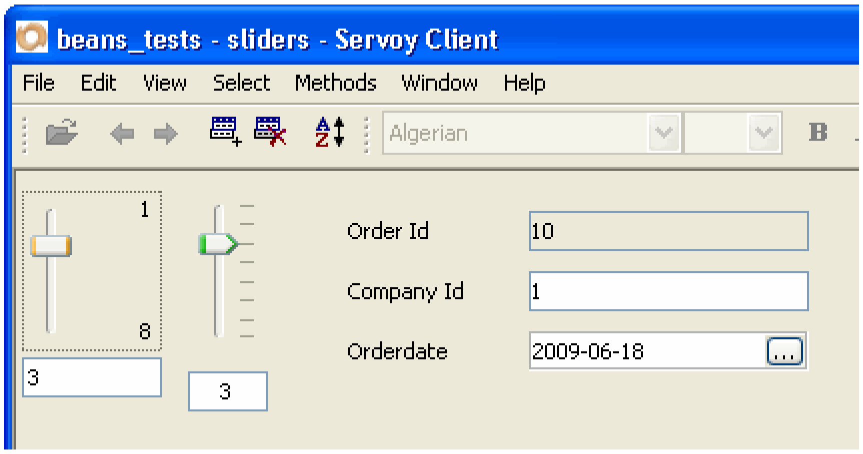
That’s great, we’re getting somewhere!
One cool thing would be to add a field to show the current value, so let’s first create a form variable to hold it and let’s call it “index”:

```
var index = 0;
```

and change our REC_selectRecord method slightly to also update its value, like this:

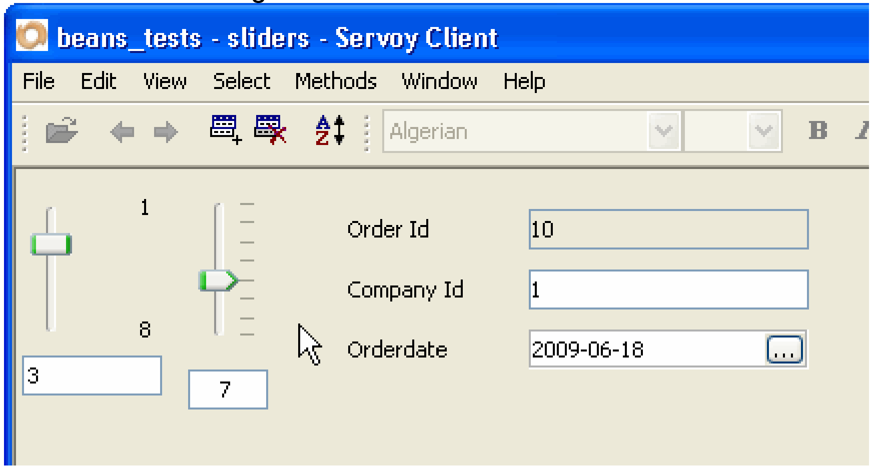
```
function REC_selectRecord()
{
	index = controller.getSelectedIndex()
	elements.slider.setValue(index);
}
```

Let’s place a field bound to this form’s variable and place it under our slider, our form should now look like this:


beans_tests - sliders - Servoy Client
File Edit View Select Methods Window Help
Algerian
B
1
8
3
3

Order Id: 10
Company Id: 1
Orderdate: 2009-06-18

Cool! But even if the values of the bean and field are properly updated when we change records, the inverse is still not true, we can move our slider’s knob or change the value of our field, the selected record doesn’t change:


beans_tests - sliders - Servoy Client
File Edit View Select Methods Window Help
Algerian
B
1
8
3
7

Order Id: 10
Company Id: 1
Orderdate: 2009-06-18

To tackle this synchronization problem, we need to find a way to change the selected record when the value of the slider changes or when the field’s value changes.

For the field, it’s easy; we can add a simple method to the field’s onDataChange event, like this one:

```
function ACTION_indexChange()
{
	controller.setSelectedIndex(index);
}
```

You can see that when you type another value inside the field and hit enter or tab, or exit the field, the selected record changes, updating the standard navigator, as well as our slider (this is the effect of our REC_selectRecord hooked onto the onRecordSelection form event).

But still, our slider, if it follows the record selection, doesn't "drive" it.
This is because to be able to react to changes of the slider's value, we need to add a **listener** to it.

Listeners are universally used in any programs that need to watch for event coming from a user's interaction, they are what's powering any event-driven program, from Servoy to your web browser to your operating system. Any modern program is actually build from a "main event" loop (a concept coming from the Apple, coming from Xerox, coming from clever people who thought a program's first function is to interact with a user, and to do this, he must listens to its "master")

Now there is a huge variety of listeners in Java, if you want to have a feel of it, you can browse the javadocs for java.util.EventListener and see all the "All Known Subinterfaces" and "All Known Implementing Classes", that's only the official java listeners enclosed in your JRE.

Listeners are powerful, there are lots of them around (each of the "onXXX" method of Servoy is actually attached to a listener), they are handy and are very often the way to build powerful and reusable code, but if you don't use them properly, they can eat your resources (CPU/RAM) quite fast.

The key to use listeners is to attach them once, and once only, and to get rid of them properly when not used anymore. To be able to do this, components usually have and addXXXXListener() and a removeXXXXListener() where XXXX is the relevant listener.

So the best thing to do before adding a listener, is actually to remove it first!
If the listener has not already been added, the removeXXXXListener() method will do nothing anyway, so it's a safe method to use. But to be sure that we will only remove our own listener (for regular Servoy field components listeners are already in place), we must keep a reference to our listener, so first we will need a variable to hold that reference, like that:

```
var listener = {};
```

Note that I set this variable to be an object by using the JavaScript construct {} – If you don't do that and either simply declare the variable like this:
```
var listener;
```

or set its value to null like this:
```
var listener = null;
```

Servoy will complain later when you try to create a listener the variable is actually a String!

Now, we've seen that there are lots of listeners around, but what kind of listeners do we need to implement to react to a change of the value of the slider. A quick look to the JSlider javadocs we can see that it holds a ChangeListener, and have an addChangeListener() and a removeChangeListener method, looks like the one we need, and if we look at the javadocs of the ChangeListener interface (javax.swing.event.ChangeListener is the fully qualified name of this interface), we can see that there is only one method “stateChanged” method defined which passes a ChangeEvent object, which (according to the javadocs of this Event will simply give us a pointer to the source of the change event)

Now we have a listener variable, and we know that our listener needs to be an object that implements the interface “ChangeListener”. - You remember that an interface is only a contract that any object can implement, but you cannot instantiate an interface in Java; so how will you implement it from inside Servoy’s JavaScript?

Well, thanks to Rhino, there is a very simple “JavaScript-like” way to implement an interface, you use the “new” keyword on an interface and you pass it a JavaScript object that holds the properties/methods the interface contract is waiting for!

Let’s add this code to FORM_onShow method:

```
if (arguments[0]) {
      listener = new
Packages.javax.swing.event.ChangeListener({stateChanged:CALLBACK_changedValue});
      elements.slider.removeChangeListener(listener);
      elements.slider.addChangeListener(listener);
}
```

The test “if (arguments[0])” simply checks if it is the first time the form is shown, the event onShow is always called by Servoy with a boolean arguments, it will be true the first time the method is called, false otherwise, this is equivalent to "onLoad" but "onShow" is called AFTER the foundset was loaded.

We first initialize the listener by using the special Rhino construct new + the use of Packages.+ fully qualified name of the object we want to create, and since it is an interface, we pass it and object {} with the stateChanged method pointing to the name of a JavaScript method “CALLBACK_changedValue” that we will create next.

Then we remove the listener using the removeChangeListener (always safe practice, especially if you are in developer and switch from developer to client all the time). Then we add it using the addChangeListener method of our bean.

Now we need to create our “stateChanged” method, in Servoy we called it CALLBACK_changedValue, let’s code it like this:

```
function CALLBACK_changedValue()
{
      var evt = arguments[0];
      if (evt) {
            if (evt.getSource() == elements.slider) {
                  index = elements.slider.getValue();
                  ACTION_indexChange();
            }
      }
}
```

We get passed an event which will be a ChangeEvent, let’s check it’s not null, and let’s also check that our source is the bean (we might use this same method later for other sources and do different things). Now if we have an event and the source is indeed our slider bean, then it means that its value has changed, all we need to do is check it, set our index to it (it will update our text field) and call our ACTION_indexChange() method which if you recall, is updating the selected record.

Why not calling controller.setSelectedRecord(index) directly in here?
Well this would mean a repetition of the same code, and would violate the DRY principle (Don't Repeat Yourself). If you already have a piece of code implementing some functionality, don't reproduce its implementation somewhere, call it instead. In short: copy/paste is hell. If you want to live long as a programmer, don't even think of using it, one day or another you would deeply regret it/

So that's our listener and callback method all done, let's just wrap the whole thing by adding a last piece of code to make sure we properly dispose of our listener, by adding a method to our form "onUnload" like this:

```
function FORM_onUnLoad()
{
	elements.slider.removeChangeListener(listener);
	listener = null;
}
```

We remove our listener for good (otherwise it might never be disposed by the Java garbage collector) from the bean listeners inner list, and set it to null.

That's it! Now to wrap it, we can remove the standard "default" navigator by setting its property to -none- and our form is now functional with a custom slider navigator…

For the lazy ones and those who are never quite sure they did everything right, the complete sample solution with all the code we have seen here is available at the url:
http://www.servoy-stuff.net/tutorials/utils/t02/beans_tests.servoy
(Rename to "beans_tests.servoy" if your browser insists that this is a zip file – and import into Servoy)

For those still on 3.5.x (why in hell?), I have made a similar solution, although slightly adapted (for lack of form variables and "onUnload" form event – boy how quirky this version is compared to 4.1!):
http://www.servoy-stuff.net/tutorials/utils/t02/sliders35.servoy

You can test it yourself and even though the JSlider bean is not web compatible, it is working well in the smart client.

Or so it seems…

In the next parts we will take a look at some of the problems and limitations this way of using beans has and see how we can address these shortcomings by wrapping a component into a Servoy Aware bean, this time using Servoy as a Java IDE…

In the meantime have fun with standard swing beans and java listeners!

Patrick Talbot
Servoy Stuff
2009-06-29