NOV
29
2013

2
COMMENTS

SERVOY TUTORIAL: USING CSS UI COMPONENTS WITH CALLBACKS

This ***Servoy tutorial*** demonstrates how to use CSS (Cascading Style Sheet) UI components in Servoy layouts, and how to capture callbacks from them so you can fire events events in Servoy.

Servoy Tutorial Photo Credit: Marco Escobedo Art / Design via Compfight

If you do some research on the Internet, you will see that they are building some amazing web controls using pure CSS. Web developers are using CSS to build gorgeous buttons, menus, tab panels, etc. Wouldn't it be sweet if we could use these CSS controls in our Web Client solutions?

Well, who said we couldn't use CSS controls on our Servoy forms? I don't think anyone said that. But Gary, how do we actually get CSS UI components to work seamlessly with Servoy? Ah, I'm glad you asked that; this is what this Servoy tutorial is all about. I am going to show you two examples of how to do it, one using an external reference to the CSS style sheet, and the other using in-line CSS. I'm also going to show you in this tutorial how to capture the onclick event from the CSS control, and generate a callback to Servoy using the WebClientUtils plugin. If you are not familiar with this plugin, head over to Servoy Forge and grab it now; you need if for this Servoy tutorial, and we will be using it in future tutorials I have planned.

Okay, so let's start out with something we need in every application, a menu. If you look around the web there are kinds of styles and variations of CSS menus; some free and some commercial. For this tutorial I chose a vertical CSS3 menu that is kind of basic, but free, and it does have the pop-out child menus that I wanted. It will be more than good enough to drive home the point of this Servoy tutorial.

Keep in mind that we are keeping this tutorial basic; we are going to do everything in the HTML so you can see what is happening. In the next tutorial, we will do it differently, so don't jump ahead and start emailing me suggestions.

To get started, I created two globals that I will use to hold some HTML. I use a global because I want to retain the HTML, even when Servoy refreshes the page, like with the onShow event. I intend to load the global with the HTML during the forms onLoad event. Yes, this type of global is an excellent thing to put into a cache that you reference by key, as demonstrated in the Servoy tutorial "Using an Object as a Cache".

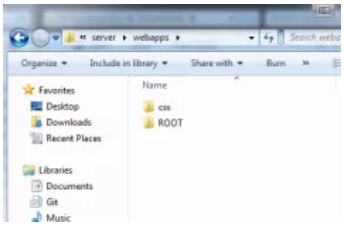
JavaScript

```
1 var _html_menu = '';
2 var _html_tabs = '';
```

On the form that I will be using the control, I create a loadHTML function and set the form's onLoad event to call it. In this function, I am building the HTML that will do all the work, like showing the CSS menu, capturing the onclick event from the menu, and generating the Servoy callback via the WebClientUtils plugin. It's really very basic HTML, so look it over carefully, as we step through what is happening here. You may need to pop-up the code into a new window so you can read it all.

In the <head> I embed a Javascript function called "runMenuHandler" using the <script> tag. The function will receive a parameter called "eventName", and pass that as an argument to the WebClientUtils callback method called "runMenuHandlerServoy". Just copy this snippet and modify it for your own scenarios.

Next I embed a link that points to the folder that contains the CSS style sheet for my menu. It is located in the Servoy application server's webapps folder. All web client resources like JQuery, CSS, etc., should be located in this folder so the Web Client can access them.



Next I add the <body> HTML tag for adding the menu, which is basically nothing more than an unordered list that uses the menu CSS. To each menu entry I added an ID and pass that to the "runMenuHandler" call in the onclick event.

JavaScript

```
1 function loadHTML() {
2
3     scopes.globals._html_menu =
4     '<html>'
5     +'<head> '
6     +'<script type="text/javascript" charset="utf-8">'
7     +'function runMenuHandler(eventName){'
8     +       plugins.WebClientUtils.generateCallbackScript(runMenuHandlerServoy, ["eventNam
9     +'}'
```

SUBSCRIBE

Be notified when new articles are released.

E-Mail

Submit

TOP POSTS & PAGES


Servoy Tutorial: Using Git Flow and SourceTree


My Insights


Servoy Tutorial: Button Magic


My Work


My Services

RECENT POSTS

Servoy Tutorial: Decorator Design Pattern – OOP

Servoy Tutorial: Factory Design Pattern – OOP

Servoy Tutorial: Button Magic

Servoy Tutorial: Table Tree View

Servoy Tutorial: Optimized Table Shuffle

Servoy Tutorial: Maintainable Code

Servoy Tutorial: Event-Driven Architecture

Servoy Tutorial: Encapsulation

Servoy Tutorial: Using Git Flow and SourceTree

Servoy Tutorial: Automated Testing

Servoy Tutorial: Coding Efficiency

Servoy Tutorial: The Mighty Array

RECENT COMMENTS

JAMES DOONAN ON **Servoy Tutorial: Button Magic**

GARY DOTZLAW ON **Servoy Tutorial: Button Magic**

JAMES DOONAN ON **Servoy Tutorial: Button Magic**

GARY DOTZLAW ON **Servoy Tutorial: The Demise of the TreeView**

```
    +'</script>'
    +'</head> '
    +'<!-- Start css3menu HEAD section --> '
    +'<link rel="stylesheet" href="/css/CSS3 Menu_files/css3menu1/style.css" type="text/css"
    +'<!-- End css3menu HEAD section --> '
    +'<body> '
    +'<!-- Start css3menu BODY section --> '
    +'<ul id="css3menu1" class="topmenu"> '
    +'    <li class="topfirst"><a id="Btn1" onclick="runMenuHandler(id)" style="width:52px;"
    +'    <li class="topmenu"><a id="Btn2" onclick="runMenuHandler(id)" style="width:52p
    +'    <ul> '
    +'        <li class="subfirst"><a id="Item 2 0" onclick="runMenuHandler(id)" >Item 2 0<
    +'        <li><a id="Item 2 1" onclick="runMenuHandler(id)" >Item 2 1</a></li> '
    +'        <li><a id="Item 2 2" onclick="runMenuHandler(id)" >Item 2 2</a></li> '
    +'    </ul></li> '
    +'    <li class="toplast"><a id="Btn2"  onclick="runMenuHandler(id)" style="width:52px;
    +'</ul>'
    +'<!-- End css3menu BODY section --> '
    +'</body> '
    +'</html>';
}
```

So in the form onLoad event, the global "_html_menu" will get loaded with all this HTML. All that remains is to place an HTML area on my Servoy form, use the global as the dataprovider, and write the "runMenuHandlerServoy" callback method. In this example, we are just outputting to the console whenever we click on any menu or sub-menu item.

JavaScript

```
function runMenuHandlerServoy(eventName) {
    application.output("Menu selected: " + eventName);
}
```

Here is what the menu looks like when viewed in Web Client. As I click on any one of the menu items, the console outputs the item ID. Obviously, you could call any method you want; you are back in Servoy after the CSS menu has been clicked. The CSS menu is now fully integrated into your Servoy layout. Cool, right?

Okay, so CSS menus work. But what about tab panels and other CSS UI components? Sure, they work the same way. If you have ever been bored with the native Servoy tab panels, then work through this example with me. I am going to take a simple CSS tab control and I am going to use it in Servoy, changing out Servoy forms as the user clicks on the CSS control. Ready? Here we go.

Exact same method as before; use a global to hold the HTML, put the text area onto the layout, set the dataprovider to the global, write the loadHTML function for the onLoad event, and finally write a callback method to process the tab ID clicked on by the user.

This time, instead of referencing external CSS, I embed the CSS directly in the HTML. It's another way of doing it and works perfectly fine. Personally, I prefer to reference the external CSS in the webapps folder for a number of reasons:

- It makes for less complicated looking HTML.
- Most CSS UI components are way more complicated then this simple one and have massive style sheets.
- You can preload it once on application start-up, and then access it from anywhere in your entire application (as you will learn in a future tutorial).

In the <body> tag, you can see where I use the same technique to embed an ID for each tab, and pass that in a "runTabHandler" method in the onclick event. This method will in turn pass the parameter to the Servoy "runTabHandlerServoy" callback method. Yes, we will learn better ways of doing all of this in the future; stop jumping ahead.

JavaScript

```
function loadHTML() {
    scopes.globals._html_tabs =
    '<html>'
    +'<head> '
    +'<script type="text/javascript" charset="utf-8">'
    +'function runTabHandler(eventName){'
    +        plugins.WebClientUtils.generateCallbackScript(runTabHandlerServoy, ["eventNam
    +'}'
    +'</script>'
    +'</head> '
    +'<!-- Start css3menu.com HEAD section --> '
    +'<link rel="stylesheet" href="/css/CSS3 Menu_files/css3menu1/style.css" type="text/cs
    +'<!-- End css3menu.com HEAD section --> '
    +'<!-- Start TAB STYLE section --> '
    +'<style type="text/css" media="screen"> '
```

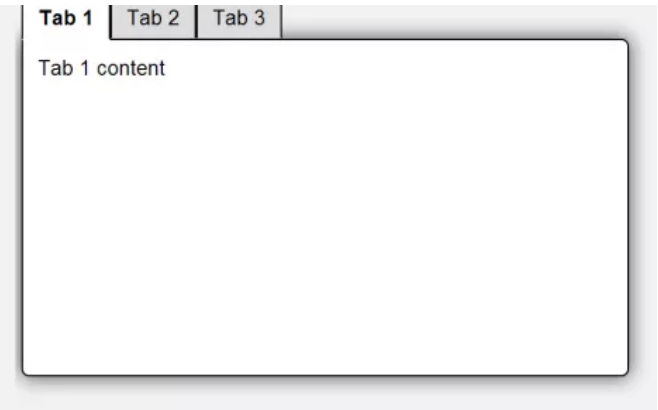
JAMES DOONAN ON **Servoy Tutorial: The Demise of the TreeView**

ARCHIVE

January 2014 (3)
December 2013 (10)
November 2013 (13)

```
+'/* Set the size and font of the tab widget */ '
+'    height: 200px; '
+'} '

+'/* Configure the radio buttons to hide off screen */ '
+'.tabGroup > input[type="radio"] { '
+'    position: absolute; '
+'    left:-100px; '
+'    top:-100px; '
+'} '

+'/* Configure labels to look like tabs */ '
+'.tabGroup > input[type="radio"] + label { '
+'    /* inline-block such that the label can be given dimensions */ '
+'    display: inline-block; '

+'    /* A nice curved border around the tab */ '
+'    border: 1px solid black; '
+'    border-radius: 5px 5px 0 0; '
+'    -moz-border-radius: 5px 5px 0 0; '
+'    -webkit-border-radius: 5px 5px 0 0; '

+'    /* the bottom border is handled by the tab content div */ '
+'    border-bottom: 0; '

+'    /* Padding around tab text */ '
+'    padding: 5px 10px; '

+'    /* Set the background color to default gray (non-selected tab) */ '
+'    background-color:#ddd; '
+'} '

+'/* Focused tabs need to be highlighted as such */ '
+'.tabGroup > input[type="radio"]:focus + label { '
+'    border:1px dashed black; '
+'} '

+'/* Checked tabs must be white with the bottom border removed */ '
+'.tabGroup > input[type="radio"]:checked + label { '
+'    background-color:white; '
+'    font-weight: bold; '
+'    border-bottom: 1px solid white; '
+'    margin-bottom: -1px; '
+'} '

+'/* The tab content must fill the widgets size and have a nice border */ '
+'.tabGroup > div { '
+'    display: none; '
+'    border: 1px solid black; '
+'    background-color: white; '
+'    padding: 10px 10px; '
+'    height: 100%; '
+'    overflow: auto; '

+'    box-shadow: 0 0 20px #444; '
+'    -moz-box-shadow: 0 0 20px #444; '
+'   -webkit-box-shadow: 0 0 20px #444; '

+'    border-radius: 0 5px 5px 5px; '
+'    -moz-border-radius: 0 5px 5px 5px; '
+'    -webkit-border-radius: 0 5px 5px 5px; '
+'} '

+'/* This matchs tabs displaying to thier associated radio inputs */ '
+'.tab1:checked ~ .tab1, .tab2:checked ~ .tab2, .tab3:checked ~ .tab3 { '
+'    display: block; '
+'} '

+'</style> '
+'<!-- End TAB STYLE section --> '

+'<body> '
+'<!-- Start TAB BODY section --> '
+'<div class="tabGroup"> '
+'    <input type="radio" name="tabGroup1" id="Tab1" onclick="runTabHandler(id)" clas
+'    <label for="Tab1">Tab 1</label> '

+'    <input type="radio" name="tabGroup1" id="Tab2" onclick="runTabHandler(id)" clas
+'    <label for="Tab2">Tab 2</label> '

+'    <input type="radio" name="tabGroup1" id="Tab3" onclick="runTabHandler(id)" clas
+'    <label for="Tab3">Tab 3</label> '

+'    <br/> '

+'    <div class="tab1">Tab 1 content</div> '
+'    <div class="tab2">Tab 2 content</div> '
+'    <div class="tab3">Tab 3 content</div> '
+'</div> '
+'<!-- End TAB BODY section --> '

+'</body> '
+'</html>';
}
```

Here is what the CSS tab panel looks like when I render the page in Web Client. Pretty nice, right? Well, there are much nicer ones out there, but it would have been more than we needed for this example. I do like the drop



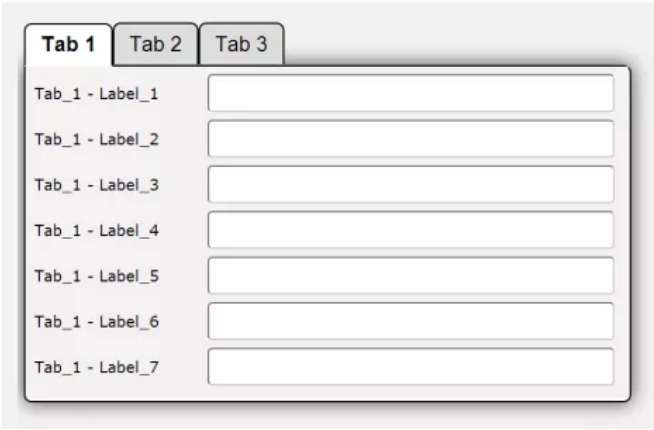
Once I had my CSS tab panel rendering fine, I placed a Servoy tab panel over top of the HTML area. I used a green background so I could align the Servoy panel perfectly over top of my HTML area. The reason for doing this, is so that I will be able to swap out Servoy forms on the tab panel when the user clicks different tabs in the CSS control.

Once I had it positioned nicely, I set the Servoy tab panel to transparent, and I wrote my "runTabHandlerServoy" callback method. Basically, all I need is simple logic to put the right Servoy form onto the tab based on the ID received in the "eventName" parameter.

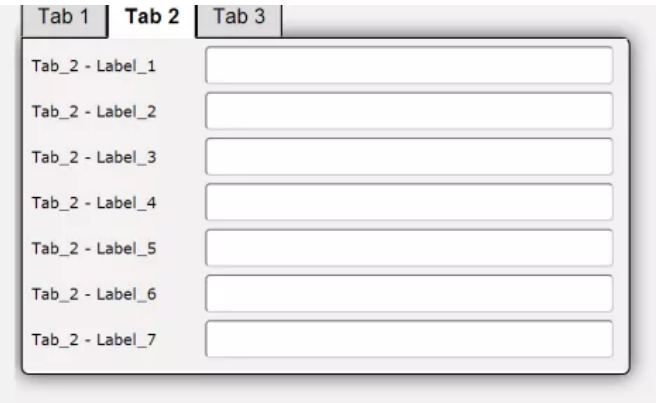
JavaScript

```
function runTabHandlerServoy(eventName) {
    application.output("Tab selected: " + eventName);

    if (eventName){
        elements.TabMain.removeAllTabs();
        if (eventName === "Tab2"){
            elements.TabMain.addTab("testCSS_tab2");
        }else if (eventName === "Tab3"){
            elements.TabMain.addTab("testCSS_tab3");
        }else{
            elements.TabMain.addTab("testCSS_tab1");
        }
    }
}
```

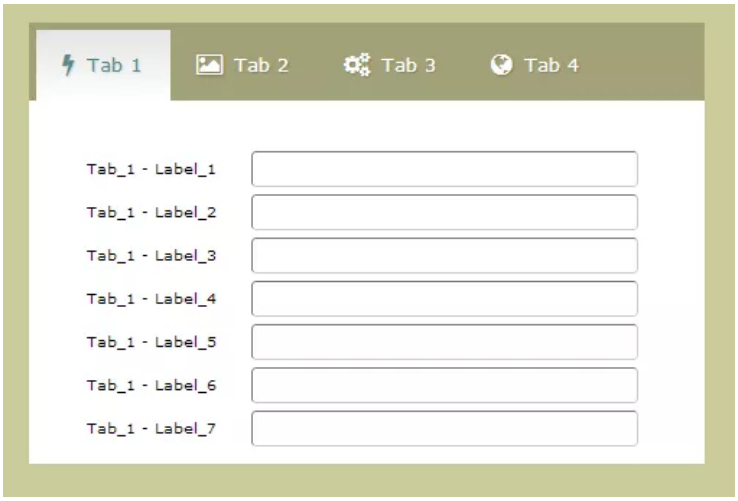
I created the three Servoy forms, and changed the labels on each so it would easily demonstrate that the switch was happening. I placed the first form onto the Servoy tab panel, so that when the Web Client loads the form, we are looking at the right Servoy form with the CSS control on "Tab 1".



Click “Tab 2” on the CSS control and the right Servoy form is instantly shown on the tab panel, giving the illusion that it is part of the CSS control itself. Works perfectly! Wasn’t that easy? Now we have a new



Okay, so maybe you think that doesn’t look that much better than native Servoy controls. Well, my friends, we did what we did to keep it simple. If you want to see what is possible, here are some screenshots of CSS control configurations I have used to wet your appetite.



This is one commercial tab control, with five different themes, and one commercial menu with a few color schemes (I can’t really demonstrate the effects , but believe me, they are numerous and very impressive).

Now everyone say again “ooohhhh….sexy!”

Alright, so there you have it. A Servoy tutorial on how to use CSS UI components with callbacks in Servoy. Using this technique you can take pretty much any CSS control and integrate it tightly into your Web Client application, making gorgeous interfaces that will knock your boss’s socks off. Maybe its time to approach him about that raise again?

I’m going to build on this technique in a future Servoy tutorial and explore how to use JQuery and DHTML components in Servoy, which I am sure will further enlighten you. There is nothing cooler than navigating to a Servoy form that shows a fully interactive, commercial grade, high-quality chart, schedule, or Gantt, completely integrated with your Servoy application.

That concludes this Servoy tutorial. I hope you enjoyed it!

Related Posts

Servoy Tutorial: The Demise of the TreeView
Servoy Tutorial: Button Magic

Bio | Latest Posts

Gary Dotzlaw

CEO at Dotzlaw Consulting

Gary Dotzlaw has 20+ years of experience as a professional software developer and has worked with over 100 companies throughout the USA and Canada. Gary has extensive qualifications in all facets of the project life cycle, including initial feasibility analysis, conceptual design, technical design, development, implementation and deployment. Gary is an expert Servoy Developer, with extensive large-commercial project experience, and has written numerous advanced tutorials on Servoy development.

2 Comments

DHANABATI
JUL 10, 2014 - REPLY

Nice article, thanks alot for sharing. Can you please provide video tutorial

GARY DOTZLAW
JUL 10, 2014 - REPLY

Thank you. I wish I had more time to work on articles and videos, but I am so busy these days with client work, that I have been unable to. I have a long list of ideas for more, but will have to wait until I get into a slow period were I can sit down and write more. In the meantime, enjoy what is here. Hope it helps.

Have something to add?

Make sure you fill in all mandatory fields.

Enter your comment here...