

## **DEVELOPING A CUSTOM SUBFILE WIDGET**

The HATS Subfile component can be used to recognize subfiles on the host screen. A scenario may be encountered in which there are multiple actions for a subfile on the host screen. We can render such a subfile by using HATS' subfile widgets.

### **USING THE HATS SUBFILE WIDGET**

By using the already provided HATS subfile widgets, we can use pop-up boxes for displaying multiple actions associated with a subfile on the webpage transformation, but it will always display a pop-up box regardless of whether there's a single action or multiple actions associated with a subfile. This can be a problem especially if subfiles for the whole application are rendered using the same widget.

### **THE NEED FOR A CUSTOM SUBFILE WIDGET**

We need a widget that can recognize the number of actions on any subfile and act accordingly to the number of actions found. It should be able to distinguish between a single action and a multiple action subfile.

### **DEVELOPING A CUSTOM WIDGET**

To overcome this roadblock in the default subfile widget, we can develop a custom subfile widget, that renders the subfile such that in the web transformation, a user may click any item in a subfile to view the actions associated with it in a pop-up box, only that the pop-up box gets displayed in the case of multiple actions. If there's a single action, the action is directly performed upon clicking an item on the webpage. We have also included scrollbars for our custom subfile widget. The addition of scrollbars helps in the navigation of different screens on the host. In the remainder of this article, we highlight the development of such a widget.

Our subfile on the host screen appears as below:

```

                                General
                                Position to . . . : _____

2=Change account      3=user      4=Delete
5=Display             7 report    8=Add

Opt      Account Cur  Account Name      Additional Information
- 0000000000001 .000 TEST
- 0000000000001 .000 TEST
- 0000000000001 .000 TEST
- 0000000000001 .000 TELLER
- 0000000000001 .001 TELLER
- 0000000000001 .002 TELLER
- 0000000000001 .003 TELLER
- 0000010000001 .004 TELLER
- 0000000000001 .005 TELLER
- 0000000000001 .004 TELLER
More...

F3=Exit  F5=Refresh  F6=Create  F12=Cancel  F13=ID
F17=Subset  F23=More options

```

As we can observe, we have multiple actions for our subfile in the above host screen. Our objective is to render the above subfile as such, that it displays the results as hyperlinks and upon clicking any hyperlink, a list of available actions is displayed to the user in a pop-up window.

Our webpage rendering appears as shown below:

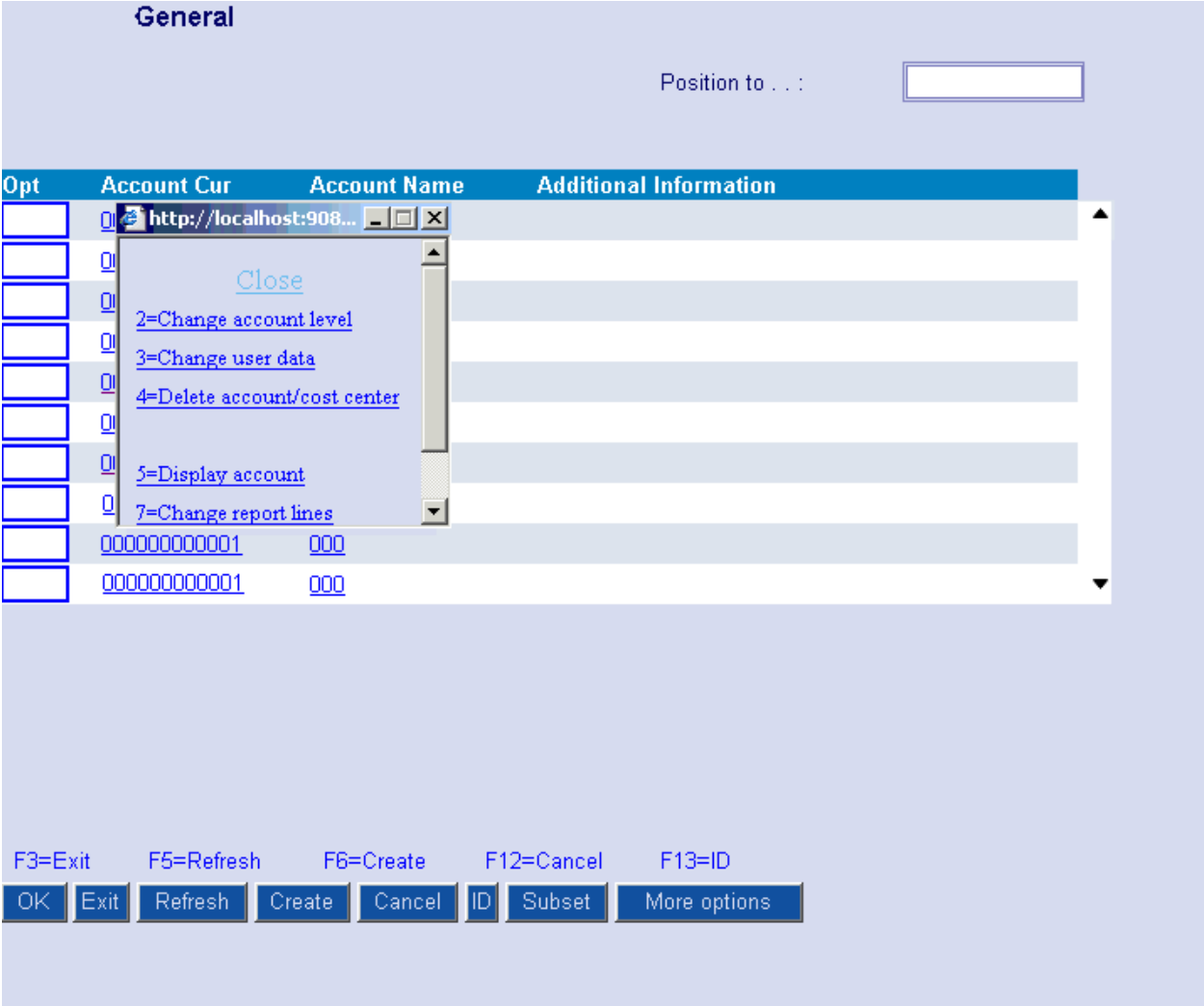
**General**

Position to . . . :

Opt	Account Cur	Account Name	Additional Information
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">001</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	
<input type="checkbox"/>	<a href="#">000000000001</a>	<a href="#">000</a>	

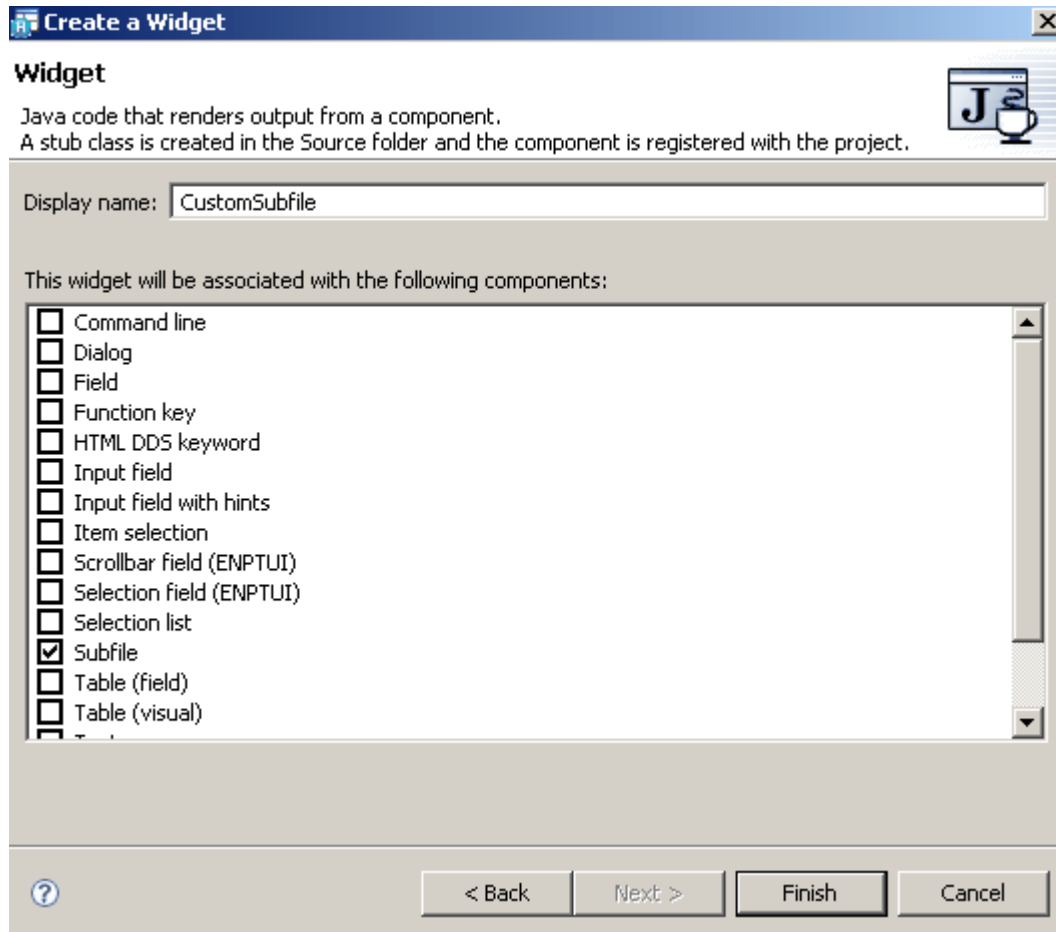
F3=Exit    F5=Refresh    F6=Create    F12=Cancel    F13=ID

Upon clicking the first link in the above screen, we are presented with a pop-up box that displays all the available actions on the host screen, as shown below:



Upon clicking any of the actions in the above screenshot, the proper action code gets populated into the respective input field and an **Enter** command is sent to the host, thus completing the action. Notice the scrollbars in the shot above, clicking the down arrow, sends a **[PAGEDOWN]** to the host and thus displays the new items on the webpage from the host screen. Similarly, clicking the up arrow sends the **[PAGEUP]** key to the host.

We have associated our widget on the Subfile Component, as shown below:



We fetch the elements into **FieldComponents** as demonstrated by the following code segment:

```
ComponentElement[] elements = this.getComponentElements();  
ComponentElementList cel = (ComponentElementList) elements[0];  
FieldComponentElement fce= new FieldComponentElement();
```

As we know that a subfile is mainly composed of **Actions**, **Header**, and **Data**. We now utilize the three components separately in our widget.

We start with the **Actions** associated with the Subfile. Let's examine the code segment below:

```
if(cel.getElement(0).getClass().toString().indexOf("SubfileActionComponentElementV6")>0)  
{  
    Object objj =cel.getElement(j);  
    if(objj instanceof SubfileActionComponentElementV6){  
        SubfileActionComponentElementV6 obj =(SubfileActionComponentElementV6) objj;  
        numActions++;  
        actions[j]=obj.getFullCaption().trim();  
    }  
}
```

We get the elements of our **cel** object in an Object **objj**. Next, we parse **objj** into a **SubfileActionComponentElementV6** object named **obj**. This object will contain our subfile **Actions**. The **numActions** variable will contain the number of **Actions** available in our subfile.

Next we place a check for a single or multiple actions on the host. In the code segment below

```
if (numActions > 1) {
    buffer = getMultipleActionSubFile (buffer, cel, fce);
} else if (numActions == 1) {
    buffer = getSingleActionSubFile (buffer, cel, fce);
}
```

We check the value of **numActions**, if its value is greater than 1, this means there are multiple actions in our subfile. Therefore we call our custom method named **getMultipleActionSubfile** that contains the functionality to render the data on the webpage as displayed earlier in the document. It populates a pop-up window and implements the functionality of sending the appropriate action through the pop-up window. Similarly, if **numActions** is equal to 1, only a single action is found, and we call our custom method **getSingleActionSubfile** that doesn't call a pop-up window and directly sends the action to the host.

Next, we can print out the **Header** element of our subfile, utilizing the **SubfileHeaderComponentElementV6** class in the code segment below:

```
buffer.append("<TR class='HATSTABLEHEADER'>");
SubfileHeaderComponentElementV6 shce = (SubfileHeaderComponentElementV6) cel.getElement(i);
```

Now we print our subfile **Data** on our webpage utilizing the **SubfileDataComponentElementV6** class to parse our **cel** object as depicted by the code segment below:

```
if (cel.getElement(i).getClass().toString().indexOf("SubfileDataComponentElementV6") > 0)
{
    counter++;
    SubfileDataComponentElementV6 sace = (SubfileDataComponentElementV6) cel.getElement(i);
    if (flag == true)
    {
        buffer.append("<TR class='HATSTABLEODDROW'>");
        flag = false;
    }
    else
    {
        buffer.append("<TR class='HATSTABLEEVENROW' align='left'>");
        flag = true;
    }
}
```

Notice that we have used different classes to highlight our odd and even rows (namely **HATSTABLEODDROW** and **HATSTABLEEVENROW**).

The following code segment calls the JavaScript function that is responsible for calling the pop-up window upon clicking of an item in the subfile **Data** section on the webpage:

```
if(!fce.isAllBlank() && fce.isAlphaNumeric() && !fce.getText().equals(""))
{
    buffer.append("<td><a href=\"javascript:checkMyValue('"+str+"');\">"+fce.getText()+ "</a>");
    buffer.append("</td>");
}
```

In this way we can design a subfile widget that can render subfiles in an application depending on the number of actions and having scrollbars for navigating to different screens on the host.



© Copyright IBM Corporation 2010  
IBM Global Services  
Route 100  
Somers, NY 10589  
U.S.A.  
Produced in the United States of America  
08-10  
All Rights Reserved

IBM, the IBM logo, [ibm.com](http://ibm.com), Lotus®, Rational®, Tivoli®, DB2® and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [ibm.com/legal/copytrade.shtml](http://ibm.com/legal/copytrade.shtml) Other company, product and service names may be trademarks or service marks of others. The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software. This document illustrates how one organization uses IBM products. Many factors have contributed to the results and benefits described; IBM does not guarantee comparable results elsewhere.