ManagingCheckboxes

Managing Checkboxes in Datatables

There has been recent discussion on the list about using checkboxes in datatables. Perhaps the primary reason for this GUI design is to provide a set of non-exclusive selections for the user where clicking/unclicking a checkbox in a row selects/unselects a row. A typical view using this approach for row-selection will usually provide some ability for the user to select/unselect all the rows. For developers, it is also important to check that the user has checked one or more rows.

Here is an example of using a checkbox in a datatable

```
<t:dataTable renderedIfEmpty="false" cellpadding="2" cellspacing="1"
   columnClasses="leftAlignCol" headerClass="list-header"
   id="dataTable1" rowClasses="list-row-even,list-row-odd" styleClass="dataTable"
   value="#{myBackingBean.queryResult}" var="currentRow" width="100%">
   <h:column >
       <t:selectBooleanCheckbox forceId="true" id="chk" onclick="onChangeSelect(this);"
        value="#{myBackingBean.selected}" />
       <f:facet name="header">
           <h:outputText value="Selected"/>
       </f:facet>
  </h:column>
      :
    more columns
      :
</t:dataTable>
```

Notice that we use Tomahawk's checkbox as we can use **forceId** on the checkbox to render an array of checkbox ids of the form chk[0], chk[1],...,chk[n-1] where n is the number of rows rendered. We also use the **onclick** property to capture when the state of each checkbox changes. Now when the user checks/unchecks a box the onChangeSelect() javascript function is invoked.

This covers the JSF which, thanks to Tomahawk, is trivial but we still need some basic JS functions which can manage the view. In particular, we need the following functions:

- a function to find the ith checkbox
- a function to monitor a check/uncheck event (our onChangeSelect function).
- a function to determine how many checkboxes are checked
- a function to set/unset all checkboxes

Each of these are described below.

Finding the ith checkbox is actually easy as long as we have used forceid on the checkbox control. If not using Tomahawk, we need to follow the JSF approach for deconflicting ids which is to prepend the form id onto the control which will render an array of checkboxes id'd as frm1:chk[0], frm1:chk[1],..., frm1:chk[n-1].

```
/*
 * Browser-safe get object. Tries all three approaches to get an object
 * by its element Id.
 *
 * Param:                objId - String - id of element to retrieve.
 * Param:                formId - String - id of form (optional).
 * Returns:               element or null if not found.
 */
function getObj(objId, formId) {
        var fullId = objId;
        if (formId != null && formId.length > 0) {
            fullId = formId + ':' + objId;
        }
        //alert('getting object: ' + fullId);
        var elem = null;
        if (document.getElementById) {
              elem = document.getElementById(fullId);
          } else if (document.all) {
            elem = document.all[fullId];
          } else if (document.layers) {
               elem = document.layers[fullId];
          }
          return elem;
}
```

Note the use of browser safe search and that we pass the form id to this function as several of our view have more than one form.

Next we need to write our onChangeSelect function. This function is used to enable/disable other controls on the view (such as submit buttons) when one or more checkboxes are checked.

```
var cmdBtnId = 'cmdbtn';
var formId = 'frm1';
var checkBoxArrayId = 'chk';
function onChangeSelect(checkbox) {
    // Render the transfer button if one or more checkboxes are selected
    hideOrShowObject(formId, cmdBtnId, checkBoxArrayHasChecked(checkBoxArrayId));
}
```

cmsBtnId is the id associated with a command button which will be hidden if no checkboxes are checkd. Now we need the ability to work through the checkboxes and determine how many are checked.

```
/*
 * Browser-safe. Checks to see if an array of check boxes has any which are
 * checked. Boxes have ids like check[0], check[1], ... , check[n] where 'check'
 * is the base Id that has been assigned to the group.
 *
 * Param:                arrayId - String - id of element group to change
 * Returns:               boolean (true one or more checked) false (else)
 */
function checkBoxArrayHasChecked(arrayId) {
    for (i = 0; ; i++) {
        id = arrayId + '[' + i + ']';
        elem = getObj(id);
        if (elem == null) {
            break;
        } else if (elem.checked) {
           return true;
        }
    }
    return false;
}
/*
 * Browser-safe. Sets the visibility of an element using CSS visibility
 *
 * Param:                objId - String - id of element to change
 * Param:                state - boolean - true (show element) false (hide element)
 * Returns:               nothing
 */
function hideOrShowObject(formId, objId, state) {
    var elem = getObj(objId, formId);
    if (elem != null) {
        if (state) {
            elem.style.visibility = 'visible';
        } else {
            elem.style.visibility = 'hidden';
        }
    }
}
```

Now when a checkbox is checked our command button is shown otherwise it is hidden. So our basic functions are down, now we need to manage the usability factors which includes (a) check all, (b) uncheck all, and (c) setting the initial view state.

```
/*
 * Browser-safe. Check or uncheck an array of checkboxes. Boxes have ids
 * like check[0], check[1], ... , check[n] where 'check' is the base Id that
 * has been assigned to the group.
 *
 * Param:                arrayId - String - id of element group to change
 * Param:                state - boolean - true (check all elements) false (uncheck all elements)
 * Returns:               nothing
 */
function checkBoxArraySet(arrayId, state) {
    for (i = 0; ; i++) {
        id = arrayId + '[' + i + ']';
        elem = getObj(id);
        if (elem == null) {
            break;
        } else {
            elem.checked = state;
        }
    }
}
```

Tying this function to two form buttons allows the user to check or uncheck the entire array.

```
<h:panelGroup >
    <h:outputLabel styleClass="label" for="all">
        <h:outputText styleClass="label" value="Select All: "/>
    </h:outputLabel>
    <h:graphicImage onclick="setAll(true);" id="all" url="resources/add.gif" title="Select all records..."/>
    <h:outputText escape="false" value=" "/>
    <h:outputLabel styleClass="label" for="none">
        <h:outputText styleClass="label" value="Unselect All: "/>
    </h:outputLabel>
    <h:graphicImage onclick="setAll(false);" id="none" url="resources/select_none.gif" title="Unselect all
records..."/>
</h:panelGroup>
```

this is the JS which performs the function

```
function setAll(state) {
    // Set the checkBox array on or off
    checkBoxArraySet(checkBoxArrayId, state);
    // Render the transfer button if one or more checkboxes are selected
    hideOrShowObject(formId, cmdBtnId, checkBoxArrayHasChecked(checkBoxArrayId));
}
```

The last task is to set the initial form state which is can be performed as an onLoad event

```
        <body onload="setInitialState()" >
```

and somewhere accessible within the page

```
// Set the cmd button off until a checkbox has been changed
function setInitialState() {
    hideOrShowObject(formId, cmdBtnId, false);
}
```