SCC
INFORMATIONSSYSTEME

COMMON CONTROLS

Events and callback methods

How a control element passes its events on to the application

SCC Informationssysteme GmbH

Version 1.6.0 · As of: 18. Juni 2006

Impressum

PUBLISHED BY

SCC Informationssysteme GmbH
Drosselweg 13
64367 Mühltal
+49 6151 13 63 10
www.scc-gmbh.com

Contents

1 Overview

This document describes how callback methods for handling user interface events are implemented within the Common Controls framework.

The framework distinguishes between callback methods for control elements and those for form elements. For both kinds, callback methods can be implemented within a Struts action class.

Important: event handling in the Common Controls only works with Struts action classes that implement the interface com.cc.framework.adapter.struts.FrameworkAction.

Beyond that, events of a control element can also be handled by the control element instance itself.

The difference between the two kinds of event is that events of control elements

Events of control elements

For an event of a control element, the callback method always receives the ControlActionContext first, followed by the parameters belonging to the event itself; callback methods of forms receive the FormActionContext instead.

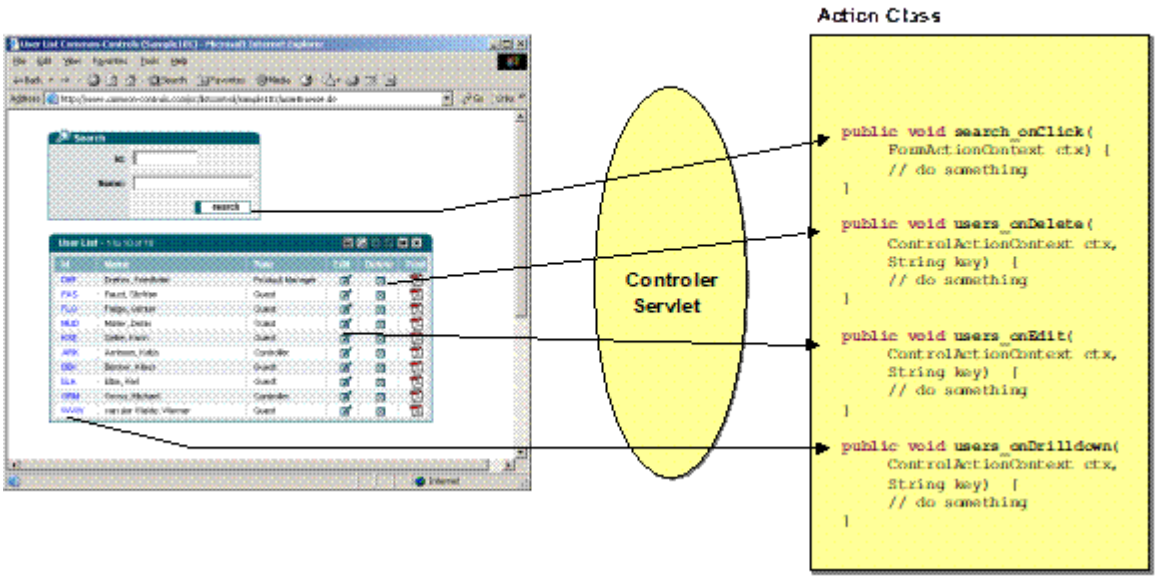
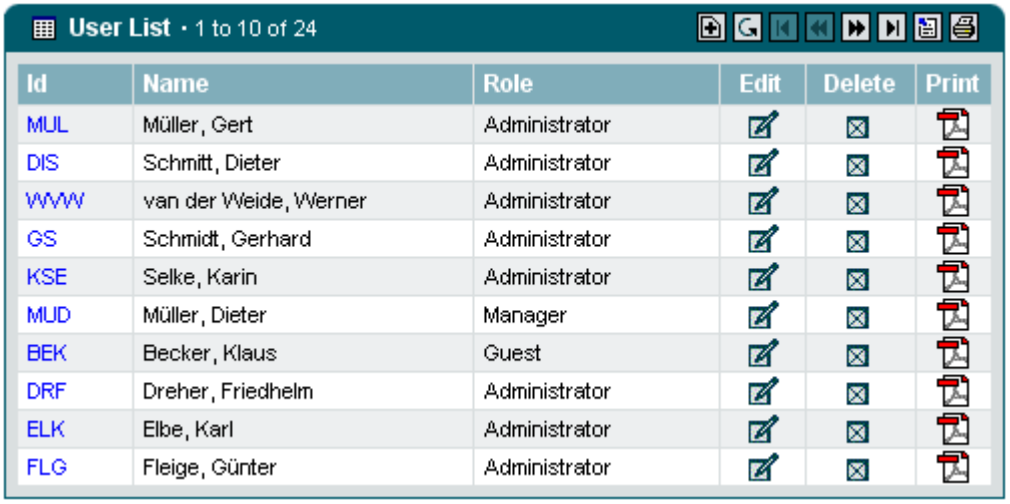


Figure 1: Callback methods for control and form elements

ControlActionContext and FormActionContext are interfaces that essentially encapsulate access to objects such as the session, request and response objects. They also provide further information about the event that occurred — a reference to the control element that raised it, for instance. Both interfaces are described in detail in section 5.

2 Callback methods of the control elements

2.1 ListControl



Id	Name	Role	Edit	Delete	Print
MUL	Müller, Gert	Administrator			
DIS	Schmitt, Dieter	Administrator			
WWW	van der Weide, Werner	Administrator			
GS	Schmidt, Gerhard	Administrator			
KSE	Selke, Karin	Administrator			
MUD	Müller, Dieter	Manager			
BEK	Becker, Klaus	Guest			
DRF	Dreher, Friedhelm	Administrator			
ELK	Elbe, Karl	Administrator			
FLG	Fleige, Günter	Administrator			

Figure 2: ListControl example

The ListControl raises the following events:

Raised by	Event	Signature of the callback method
Drilldown column	Drilldown	CtrlName_onDrilldown(ControlActionContext ctx, String key) throws Exception
Edit column	Edit	CtrlName_onEdit(ControlActionContext ctx, String key) throws Exception
Delete column	Delete	CtrlName_onDelete(ControlActionContext ctx, String key) throws Exception
Selector column	Select	CtrlName_onSelect(ControlActionContext ctx, String key) throws Exception
Button column	CellClick	CtrlName_onCellClick(ControlActionContext ctx, String column, String key) throws Exception If the "command" attribute is given, the corresponding method is called CtrlName_onCommand(ControlActionContext ctx, String key) throws Exception
Image column	CellClick	CtrlName_onCellClick(ControlActionContext ctx, String column, String key) throws Exception
Hyperlink column	CellClick	CtrlName_onCellClick(ControlActionContext ctx, String column, String key) throws Exception
Check column	Check	CtrlName_onCheck(ControlActionContext ctx, String key, SelectMode mode, boolean checked) throws Exception
Checkbox column	CheckAll	CtrlName_onCheckAll(ControlActionContext ctx, SelectMode mode, boolean checked) throws Exception
Add button	Create	CtrlName_onCreate(ControlActionContext ctx) throws Exception

Raised by	Event	Signature of the callback method
Refresh button	Refresh	CtrlName_onRefresh(ControlActionContext ctx) throws Exception
PrintButton	PrintList	CtrlName_onPrintList(ControlActionContext ctx) throws Exception
ExportButton	ExportList	CtrlName_onExportList(ControlActionContext ctx) throws Exception
Sort button	Sort	CtrlName_onSort(ControlActionContext ctx, String column, SortOrder direction) throws Exception
Turning to another page	Page	CtrlName_onPage(ControlActionContext ctx, int page) throws Exception

Table 1: Events of the ListControl

Arguments:

```
CODE

1  ctx         ControlActionContext. See section 5.1.
```

column — name of the column in which the event was raised; the property of the column is returned as the name.

```
CODE

1  direction   SortOrder. The sort direction within the column (ascending or descending)
2  key         Unique key of the row as supplied by the ListDataModel.
3  mode        SelectMode. States whether only one element may be selected in a checkbox column or
   several at the same time.
4  page        The page to jump to.
5              -1 = jump to the last page
6              0..n = jump to the given page
7              Note: the handler itself has to keep the value within the valid range. This method
   should only be implemented if a list is meant to page differently.
```

2.2 TreeControl

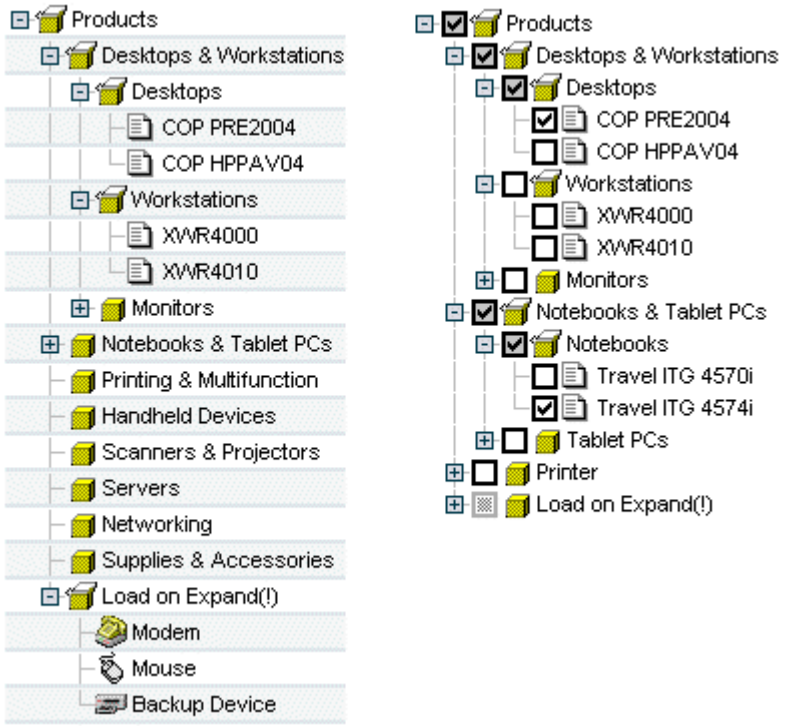


Figure 3: TreeControl example

The TreeControl raises the following events:

Raised by	Event	Signature of the callback method
Opening a group node	Expand	CtrlName_onExpand(ControlActionContext ctx, String key) throws Exception
Opening a group node with an unknown number of children	ExpandEx	CtrlName_onExpandEx(ControlActionContext ctx, String key) throws Exception
Closing a group node	Collapse	CtrlName_onCollapse(ControlActionContext ctx, String key) throws Exception
Clicking a checkbox	Check	CtrlName_onCheck(ControlActionContext ctx, String key, SelectMode mode, boolean checked) throws Exception

Table 2: Events of the TreeControl

Arguments:

```
CODE

1  ctx        ControlActionContext. See section 5.1.
```

key — unique key of the row as supplied by the TreeNodeDataModel.

```
CODE
```

```
mode        SelectMode. States whether only one element may be selected in a checkbox column or
several at the same time.
```

2.3 TreeListControl

Figure 4: TreeListControl example

(figure reduced in size)

The TreeListControl combines a tree with a list and can therefore raise every event described in the chapters on the TreeControl and the ListControl. Only the additional events are covered here.

The TreeListControl raises the following additional events:

Raised by	Event	Signature of the callback method
Add column	Add	Ctrlname_onAdd(ControlActionContext ctx, String key) throws Exception

Table 3: Events of the TreeListControl

```
CODE
```

```
ctx         ControlActionContext. See section 5.1.
```

key — unique key of the row as supplied by the TreeNodeDataModel.

2.4 TabSetControl

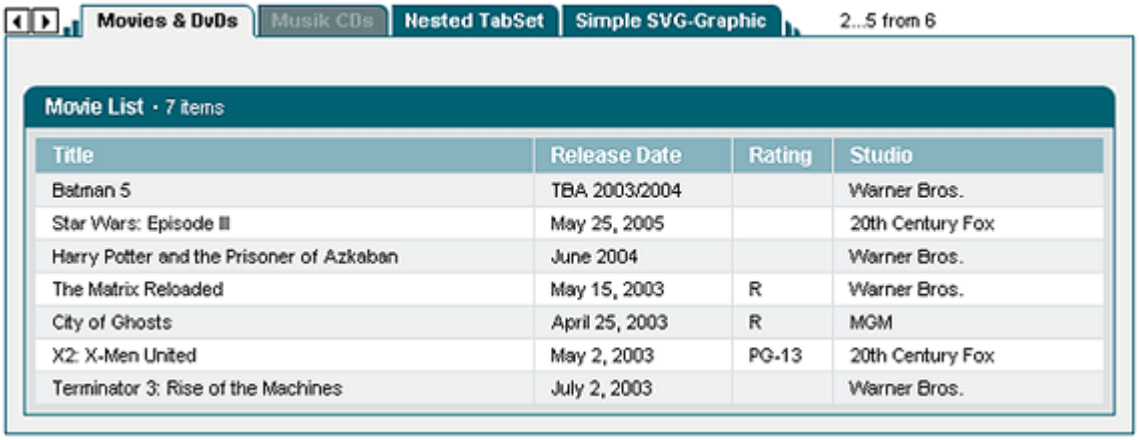


Figure 5: TabSetControl example

(figure reduced in size)

The TabSetControl raises the following events:

Raised by	Event	Signature of the callback method
TabPage	TabClick	CtrlName_onTabClick(ControlActionContext ctx, String seltab) throws Exception

Table 4: Events of the TabSetControl

Arguments:

```JSP
ctx       ControlActionContext. See section 5.1.
seltab    Id of the selected tab as assigned in the JSP page when the tab was declared. In the example below either tab1, tab2 or tab3.

<ctrl:tabset
     styleId="tabset1"
     name="maintabset"
     action="sample401/tabsetBrowse"
     tabs="6"
     labellength="20"
     width="650"
     imagemap="im_tabs"
     runat="server">

     <ctrl:tab   tabid="tab1"     action="/sample401/tabpage1"    title="Books" content="Tab_Page1.jsp"      tooltip="Books"/>
     <ctrl:tab   tabid ="tab2"    action="/sample401/tabpage2"    title="Movies & DVD's" content="Tab_Page2.jsp"      tooltip="Movies"/>
     <ctrl:tab   tabid ="tab3"    action="/sample401/tabpage3"    title="Musik CD's" content="Tab_Page3.jsp"      tooltip="Disabled Tab"   enable="false"/>
</ctrl:tabset>

```

A TabSetControl may contain further control elements.

2.5 BreadCrumbControl

Figure 6: BreadCrumbControl example

(figure reduced in size)

The BreadCrumbControl raises the following events:

Raised by	Event	Signature of the callback method
BreadCrumb	Crumb click	CtrlName_onCrumbClick(ControlActionContext ctx, String key) throws Exception

Table 5: Events of the BreadCrumbControl

Arguments:

JSP

```
ctx       ControlActionContext. See section 5.1.
key       Id of the selected crumb as assigned in the JSP page when the bread crumb was declared. In
the example below either crumb1, crumb2 or crumb3.

<menu:crumbs
     value="crumb2"
     action="crumb701/crumbBrowse"
     labellength="40"
     imagemap="im_user"
     name="crumbcrl">

     <menu:crumb     crumbid="crumb1"     action="sample510/breadcrumb"
          title="crumb1"   tooltip="crumb1"/>

     <menu:crumb     crumbid="crumb2"     action="sample510/breadcrumb"
          title="crumb2"   tooltip="crumb2"/>

     <menu:crumb     crumbid="crumb3"     action="sample510/breadcrumb"
          title="crumb3"   tooltip="crumb3"  imageref="controller"/>

</menu:crumbs>

```

2.6 SchedulerControl

Figure 7: SchedulerControl example

(figure reduced in size)

The SchedulerControl raises the following events:

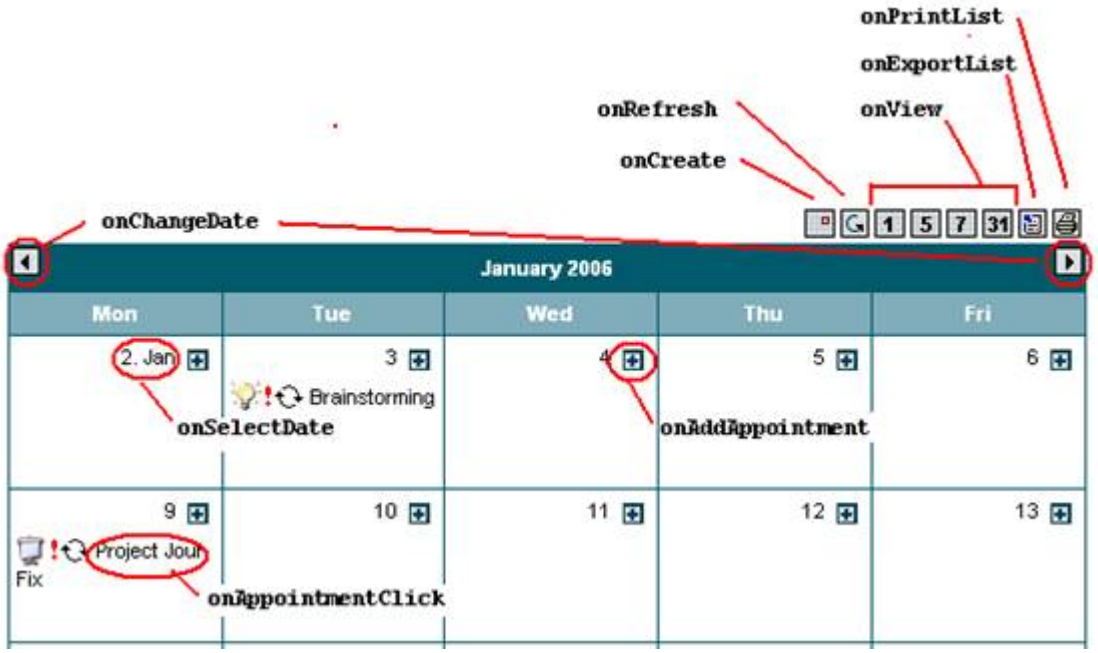
Raised by	Event	Signature of the callback method
Create button	Create	CtrlName_onCreate(ControlRequestContext ctx) throws Exception
Add button for adding an entry on a given day	AddApointment	CtrlName_onAddAppointment(ControlRequestContext ctx, long timeInMillis) throws Exception
Refresh button	Refresh	CtrlName_onRefresh(ControlRequestContext ctx) throws Exception
Clicking an appointment	AppointmentClick	CtrlName_onAppointmentClick(ControlRequestContext ctx, String key, long timeInMillis) throws Exception
Previous / next date	ChangeDate	CtrlName_onChangeDate(ControlRequestContext ctx, long timeInMillis, String view) throws Exception
Selecting an appointment	CheckAppointment	CtrlName_onCheckAppointment(ControlRequestContext ctx, String uniqueId, long timeInMillis, boolean check) throws Exception
Selecting a day	CheckDate	CtrlName_onCheckDate(ControlRequestContext ctx, long timeInMillis, SchedulerScope scope, boolean check) throws Exception
Branching to the detail view	SelectDate	CtrlName_onSelectDate(ControlRequestContext ctx, long timeInMillis, String view) throws Exception
View button	View	CtrlName_onView(ControlRequestContext ctx, String view) throws Exception
PrintButton	PrintList	CtrlName_onPrintList(ControlRequestContext ctx) throws Exception
ExportButton	ExportList	CtrlName_onExportList(ControlRequestContext ctx) throws Exception

Table 6: Events of the SchedulerControl

Arguments:

CODE

```
ctx             ControlRequestContext.
timeInMillis    Date.
key             Id of the appointment.
view            States which view to switch to: day, workweek, week, month, year.
uniqueId        Id of the appointment.
check           true if selected, false otherwise.
```



(checkboxes="true")

3 Callback methods of form buttons

Form buttons always raise an onClick event, which leads to a call of the corresponding callback method in the action class. The only condition is that the name of the button in the name attribute begins with the prefix “btn”.

To use hover effects, the id attribute of the button also has to be given; that id likewise has to begin with the prefix “btn”. The images for the various states have to be present as well.

The name of the callback method is always made up of the name of the button — in lower case and without the “btn” prefix — and the suffix “_onClick”. Its first parameter is always the FormActionContext.

Form buttons raise the following event:

Raised by	Event	Signature of the callback method
Button	Click	buttonname_onClick(FormActionContext ctx)

Table 7: Event of a form element

The class FormActionContext is described in chapter 5.2.

Examples:

Naming the button on the JSP page:

CODE

```
<form action="useredit.do" method="post">

    <input name="btnSave" src="images/btnSave1.gif" title="Save all Changes"/>
    <input name="btnBack" src="images/btnBack1.gif" title="Back"/>
</form>

```

or when using the Struts form tag:

JSP

```
<%@ taglib uri="/WEB-INF/tlds/struts-html.tld"  prefix="html" %>

<html:form action="useredit.do" method="post">

    <input name="btnSave" src="images/btnSave1.gif" title="Save all Changes"/>
    <input name="btnBack" src="images/btnBack1.gif" title="Back"/>
</html:form>

```

or when using both the Struts and the Common Controls form tags:

JSP

```
<html:form action="/sample101/userEdit">

    <forms:form type="edit" caption="User - Edit" formid="frmEdit">
        <forms:plaintext  label="User-Id"  property="userId"/>
        <forms:text label="First-Name"  property="lastName" size="45" required="true"/>
        <forms:text label="Last-Name" property="firstName"  size="45" required="true"/>

        <forms:select  label="Role"  property="rolekey">
            <base:options property="roleOptions"/>
        </forms:select>

        <%-- ***************** --%>
        <%-- ** Form Buttons ** --%>
        <%-- ***************** --%>
        <forms:buttonsection default="btnSave">
            <forms:button name="btnSave" src="images/btnSave1.gif"  title="Save"/>
            <forms:button name="btnBack" src="images/btnBack1.gif"  title="Back"/>
        </forms:buttonsection>

    </forms:form>
</html:form>

```

The matching callback methods in the action class:

For the buttons used on the JSP page, the following two callback methods are added to the action class:

- save_onClick(FormActionContext ctx)
- back_onClick(FormActionContext ctx)

JAVA

```
public class UserEditAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FrameworkAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx) throws Exception {

    // do something

       ctx.forwardToInput();
    }

    // ------------------------------------------------
    //  Event Handler
    // ------------------------------------------------

    /**
     * This Method is called if the Back-Button on the
     * HTML-Form is pressed.
     * @param  ctx  FormActionContext
     * @throws Exception
     */
    public void back_onClick(FormActionContext ctx) throws Exception {
         ctx.forwardByName(Forwards.BACK);
    }

    /**
     * This Method is called if the Save-Button on the
     * HTML-Page is pressed.
```

```
30       * @param       ctx FormActionContext
31       * @throws Exception
32       */
33      public void save_onClick(FormActionContext ctx) throws Exception {
34     // save changes
35 
36           ctx.forwardByName(Forwards.SUCCESS);
37      }
38 }
39 
```

Take great care over the exact spelling: otherwise the implemented method cannot be found and called through the reflection API, which raises a framework exception and leads to doExecute() being called instead.

4 How it works

How the framework passes events of a control element on to the callback methods depends on the configured property “formelement” of that control element. The distinction between the two mechanisms is transparent to the developer: the event handlers are implemented in exactly the same way either way.

4.1 formelement=“false“

Every event of the control element triggers a hyperlink carrying all the parameters needed to reach the event handler. If a control element configured this way is used inside an HTML form, all user input in that form is lost on the round trip to the server.

The generated hyperlink consists of the following parts:

Parameter	Meaning
ctrl	Name of the control element that raised the event.
action	Identifies the action the user triggered (drilldown, edit, delete and so on).
key	Key identifying the element the action is to be carried out on.

4.2 formelement=“true“

No hyperlinks are generated. Instead, the form the control element is embedded in is submitted to the server whenever the control element raises an action, and automatically generated hidden fields see to it that the right event handler is called. The advantage is that user input in the HTML form survives the round trip to the server.

4.3 Event dispatching

For an incoming request, the Struts RequestProcessor determines the action class that is to handle it. The request is then parsed by the superclass FWAction, from which the action called is derived, and it is checked whether the request came from a control element or from a form element (see figure 1).

For a control element, the matching event handler in the action class is called on the basis of the name of the control and the event raised. The name of the control element is that of the bean holding its instance.

As in Struts, the following applies:

· If the ListControl is passed directly, in the session for example, the bean name is the name of the attribute under which the bean was placed in the session.

- If the ListControl lives inside an action form, the bean name is the name of the property under which the instance of the control is held in that form.

Example of case 1:

In this example the instance of the ListControl is placed in the session.

```
JAVA
```

```java
public class UserBrowseAction extends FWAction {

     /**
      * @see com.cc.framework.adapter.struts.FrameworkAction#doExecute(ActionContext)
      */
     public void doExecute(ActionContext ctx) throws Exception {

     try {
           UserDisplayList dspData = DBUser.fetch();
           SimpleListControl userList = new SimpleListControl();
           userList.setDataModel(dspData);
           ctx.session().setAttribute("users", userList);

     } catch (Throwable t) {
           ctx.addGlobalError(Messages.ERROR, t);
     }

     // Display the Page with the UserList
           ctx.forwardToInput();
     }
}
```

On the JSP page the bean is accessed through the name attribute. In this example the control element is not embedded in a form — unlike in example 2 — so the action attribute has to be given as well; it is through this attribute that the events reach the action class.

JSP

```jsp
<ctrl:list
    action="sample101/userBrowse"
    name="users"
    title="User List"
    width="500"
    rows="10"
    refreshButton="true"
    createButton="true">

    <ctrl:columndrilldown   title="Id"      property="userId"     width="65"/>
    <ctrl:columntext        title="Name"    property="name"        width="350"/>
    <ctrl:columntext        title="Role"    property="role.value" width="150"/>
    <ctrl:columnedit        title="Edit"/>
    <ctrl:columndelete      title="Delete"/>
</ctrl:list>

```

Example of case 2:

Here the ListControl lives inside an action form. Since the ListControl has to keep track of its own state across several round trips to the server, the form is given the scope “session” in struts-config.xml. When the instance is held inside the action form, the data is initialised in the action class — using the form's method setDataModel(ListDataModel dataModel), for example.

JAVA

```java
public class UserBrowseActionForm extends FWActionForm {
    /**
     * Instance of our list control to display the user list
     */
```

```
    private SimpleListControl users = new SimpleListControl();

    /**
     * Returns the user list
     * @return The list control
     */
    public ListDataModel getUsers {
        return users.getDataModel();
    }

    /**
     * Sets the data for the user list
     * @param dataModel The datamodel for the control
     */
    public void setDataModel(ListDataModel dataModel) {
        users.setDataModel(dataModel);
    }

}
```

On the JSP page the bean is accessed through the property attribute. If that attribute is given, the control element looks for its data model automatically inside the action form assigned to the action. The instance is reached through the property named, which in our example leads to a call of getUsers().

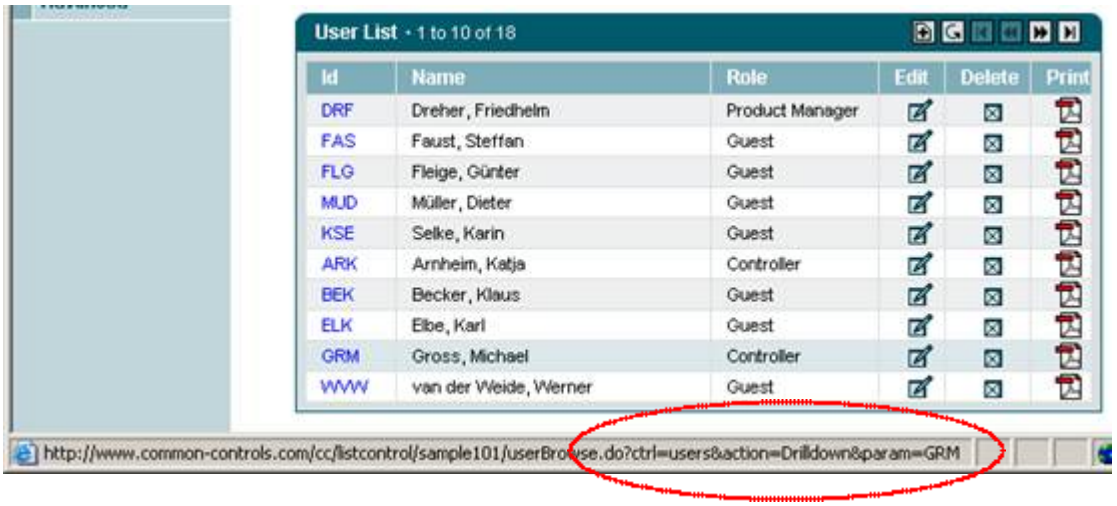
When the control element is embedded in a form, the action attribute is no longer given.

JSP

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:form action="/sample101/userBrowse">

<ctrl:list
    property="users"
    styleId="userlist1"
    title="User List"
    width="500"
    rows="10"
    refreshButton="true"
    createButton="true">

    <ctrl:columndrilldown   title="Id"      property="userId"      width="65"/>
    <ctrl:columntext        title="Name"    property="name"        width="350"/>
    <ctrl:columntext        title="Role"    property="role.value" width="150"/>
    <ctrl:columnedit        title="Edit"/>
    <ctrl:columndelete      title="Delete"/>
</ctrl:list>

</html:form/>

```

When using the property attribute, always make sure that the getter and the setter take and return the same type of object.

The name of the event handler triggered by an action on a control element can also be read off the status bar of the browser. In the following example the user clicks the record “GRM” in the column “Id” to reach the detail view.



The hyperlink generated carries the following parameters:

Parameter	Meaning	Value
ctrl	Name of the control element that raised the event.	users
action	Identifies the action the user triggered (drilldown, edit, delete and so on).	Drilldown
key	Key identifying the element a further action is to be carried out on.	GRM

The callback method in our action class is named users_onDrilldown and receives the key “GRM” alongside the ControlActionContext.

5 Classes

5.1 ControlActionContext

The interface ControlActionContext extends the interface ActionContext by the methods needed for handling events of control elements.

Through this interface the following can be done within a callback method:

Method	Description
mapping()	Gives access to the ActionMapping object.
form()	Gives access to the form bean assigned to the action in struts-config.xml.
request()	Gives access to the request object.
response()	Gives access to the response object.
session()	Gives access to the session object.
forwardToInput()	Forwards to the page named in the action mapping through its input attribute.
forwardToAction()	Forwards to the action given.
forwardByName()	Looks up an ActionForward by the name given.
getErrors()	Returns the error collection.
getMessages()	Returns the message collection.
addError()	Adds an exception to the global error collection. Note: unlike Struts, the framework carries the error and message collections through several redirects, right up to the next JSP page.
addGlobalError()	Stores a general error not tied to an input field; used to present error messages in the calling dialogue when returning from a subdialogue.
addPropertyError()	Stores an error for one property in the error collection.
addMessage()	Adds a message to the global message collection.
addGlobalMessage()	Stores a message not tied to a property; used to present messages in the calling dialogue when returning from a subdialogue.
addPropertyMessage()	Stores a message for one property in the message collection.
hasMessages()	Checks whether any messages are present.
hasErrors()	Checks whether any error messages are present.
getPrincipal()	Returns the principal object of the cc-framework security system.
control()	Returns the instance of the control element that raised the event.

Method	Description
action()	
getActionMethod()	Returns the name of the method the action handler calls once the event has occurred, in the form [controlname]_on[Action].

5.2 FormActionContext

The interface FormActionContext extends the interface ActionContext by the methods needed for handling events of buttons.

Method	Description
getActionMethod()	Returns the name of the method the action handler calls once the event has occurred, in the form [formname]_on[Action].
The remaining methods are described in section 5.1.	

5.3 SelectMode

Value	Meaning
NONE	No selection mode given.
SINGLE	Only one element may be selected.
MULTIPLE	Any number of elements may be selected.

5.4 SortOrder

Value	Meaning
NONE	No sort order given.
ASCENDING	Ascending order.
DESCENDING	Descending order.

6 Appendix

6.1 Class diagram of ActionContext