Common-Controls Events & Callback-Methods

Version 1.6.0 - Stand: 18. Juni 2006

Publisher:
SCC Informationssysteme GmbH
64367 Mühltal

Tel: +49 (0) 6151 / 13 6 31 12
Internet http://www.scc-gmbh.com

Product Site:
http://www.common-controls.com

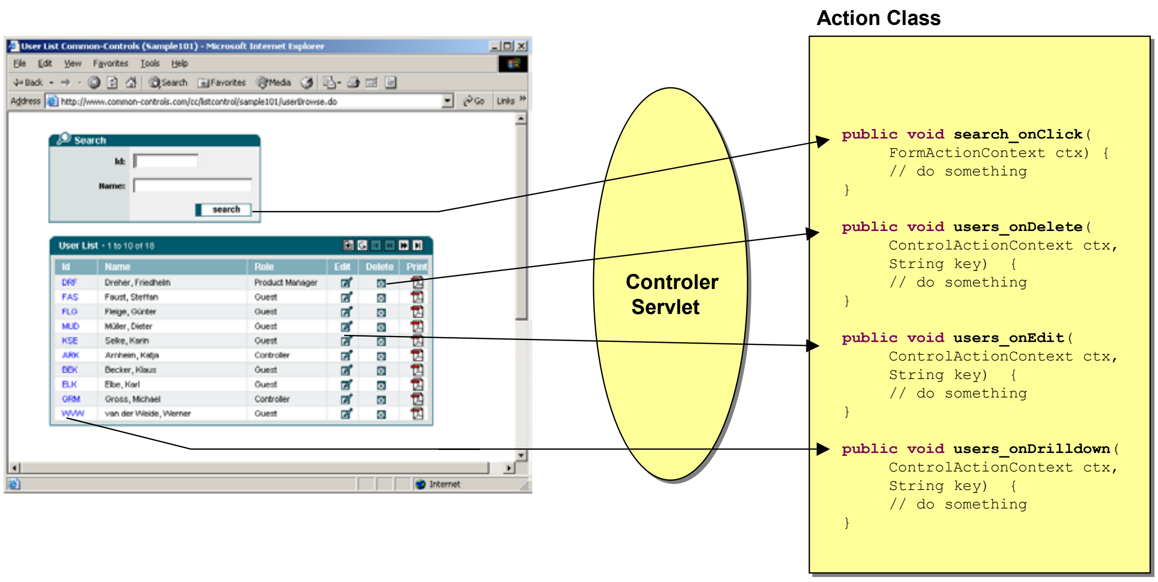
Table of contents

1 Overview

This document describes how Callback methods for handling User Interface events are implemented within the Common Controls Framework.

The Framework essentially distinguishes between Callback methods for ***control elements*** and ***form elements***. Callback methods can be implemented within an action class for both types.

In the case of an event of a control element, the Callback method – apart from other parameters – is always passed the `ControlActionContext`, whereas Callback methods of forms get the `FormActionContext` on being called.


Action Class
Controler Servlet
User List Common-Controls (Sample101) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.common-controls.com/cc/listcontrol/sample101/userBrowse.do
Go Links
Search
Id:
Name:
search
User List - 1 to 10 of 18
Internet

Id	Name	Role	Edit	Delete	Print
DRF	Dreher, Friedhelm	Product Manager			
FAS	Faust, Steffan	Guest			
FLG	Fleige, Günter	Guest			
MUD	Müller, Dieter	Guest			
KSE	Selke, Karin	Guest			
ARK	Arnheim, Katja	Controller			
BEK	Becker, Klaus	Guest			
ELK	Elbe, Karl	Guest			
GRM	Gross, Michael	Controller			
WWW	van der Weide, Werner	Guest			

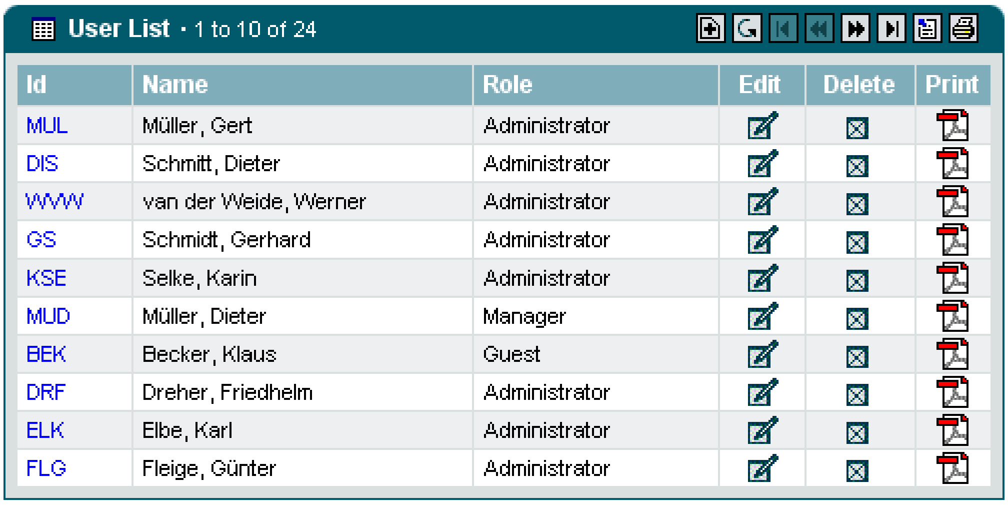
```
public void search_onClick(
      FormActionContext ctx) {
      // do something
}

public void users_onDelete(
      ControlActionContext ctx,
      String key)  {
      // do something
}

public void users_onEdit(
      ControlActionContext ctx,
      String key)  {
      // do something
}

public void users_onDrilldown(
      ControlActionContext ctx,
      String key)  {
      // do something
}
```

Figure 1: Callback methods for control and form elements

In the case of the `ControlActionContext and the FormActionContext`, what is involved are the interfaces which essentially encapsulate access to objects like the Session-object, Request-object or Response-object. In addition, they provide additional information for the event that has occurred, such as a reference to the control element that has triggered the event. Both interfaces are described in further detail in Section 5.

2 Callback Methods Controls

2.1 ListControl



User List · 1 to 10 of 24

Id	Name	Role	Edit	Delete	Print
MUL	Müller, Gert	Administrator	✎	☒	📄
DIS	Schmitt, Dieter	Administrator	✎	☒	📄
WVW	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: Example ListControl

The ListControl generates the following events:

Trigger	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 specified, the corressponding method will be called.`
		
		*Ctrlname*_**`on`***`Command`*`(`
		`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`
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`
On paging	Page	*Ctrlname*_**onPage(** `ControlActionContext ctx,` `int page) throws Exception`

Table 1: Events ListControl

Arguments:

ctx	ControlActionContext. See Section 5.1.
column	Name of the column in which the event was triggered. The property of the column is returned as the name.
direction	SortOrder. The sorting order within the column (increasing or decreasing)
key	Unique key of the row, as supplied by the ListDataModel.
mode	SelectMode. Specifies whether, in a Checkbox column, only one element is selectable, or several may be selected at the same time.
page	The page to which the system should go. -1 = jump to the last page 0..n = jump to the specified page Caution: The Handler itself must ensure adherence to the valid range. This method may only be implemented if another scrolling behavior is required in a list.

2.2 TreeControl

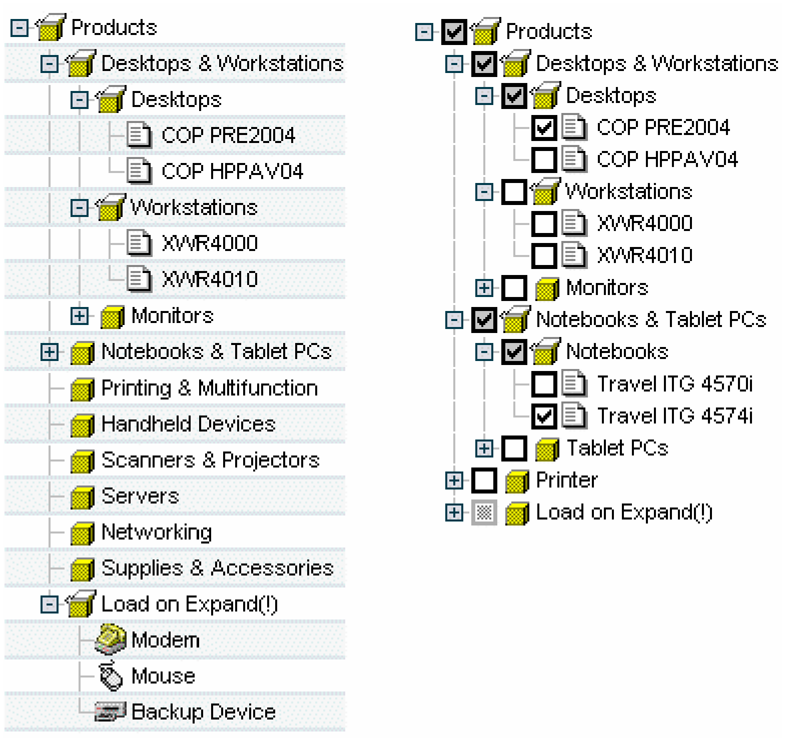


Figure 3: Example TreeControl

The TreeControl generates the following events:

Trigger	Event	Signature of the Callback Method

Open group node	Expand	*Ctrlname*_**`onExpand`**`(`
		`ControlActionContext ctx,`
		`String key)` *`throws Exception`*

Open group node with unknown number of children	ExpandEx	*Ctrlname*_**`onExpandEx`**`(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Close group node	Collapse	*Ctrlname*_**`onCollapse`**`(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

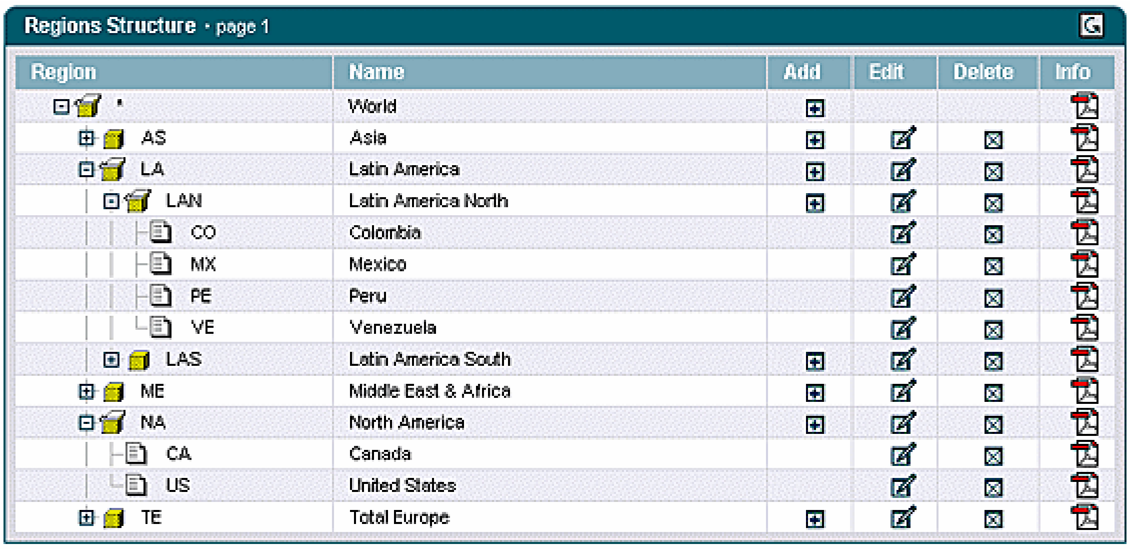
Click on checkbox	Check	*Ctrlname*_**`onCheck`**`(`
		`ControlActionContext ctx,`
		`String key,`
		`SelectMode mode,`
		`boolean checked) throws Exception`

Table 2: Events TreeControl

Arguments:

`ctx`	`ControlActionContext. See Section 5.1.`
key	`Unique key of the row as supplied by the TreeNodeDataModel.`
`mode`	`SelectMode. Specifies whether only one element is selectable in a checkbox column or several can be selected simultaneously.`

2.3 TreeListControl



Regions Structure · page 1

Region	Name	Add	Edit	Delete	Info
*	World	⊞			📄
AS	Asia	⊞	✎	☒	📄
LA	Latin America	⊞	✎	☒	📄
LAN	Latin America North	⊞	✎	☒	📄
CO	Colombia		✎	☒	📄
MX	Mexico		✎	☒	📄
PE	Peru		✎	☒	📄
VE	Venezuela		✎	☒	📄
LAS	Latin America South	⊞	✎	☒	📄
ME	Middle East & Africa	⊞	✎	☒	📄
NA	North America	⊞	✎	☒	📄
CA	Canada		✎	☒	📄
US	United States		✎	☒	📄
TE	Total Europe	⊞	✎	☒	📄

Figure 4: Example TreeListControl

(Figure reduced in size)

The TreeListControl combines a tree with a list. As a consequence, it can generate *all* the Events that are described in the chapter on the TreeControl and the ListControl. At this point, we shall only treat the events that can occur additionally.

The TreeListControl generates the following additional events:

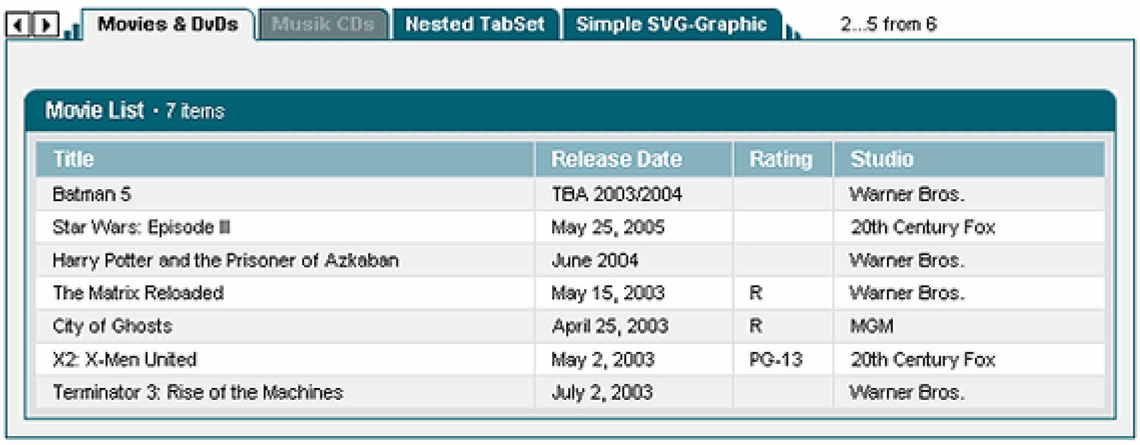
Trigger	Event	Signature of the Callback Method
Add Column	Add	*Ctrlname*_**onAdd**(ControlActionContext ctx, String key) throws Exception

Table 3: Events TreeListControl

Arguments:

ctx	`ControlActionContext. See Section 5.1.`
key	`Unique key of the row as supplied by the TreeNodeDataModel.`

2.4 TabSetControl



Title	Release Date	Rating	Studio
Batman 5	TBA 2003/2004		Warner Bros.
Star Wars: Episode III	May 25, 2005		20th Century Fox
Harry Potter and the Prisoner of Azkaban	June 2004		Warner Bros.
The Matrix Reloaded	May 15, 2003	R	Warner Bros.
City of Ghosts	April 25, 2003	R	MGM
X2: X-Men United	May 2, 2003	PG-13	20th Century Fox
Terminator 3: Rise of the Machines	July 2, 2003		Warner Bros.

Figure 5: Example TabSetControl

(Figure reduced in size)

The TabSetControl generates the following events:

Trigger	Event	Signature of the Callback Method
TabPage	TabClick	*Ctrlname*_**`onTabClick`**`( ControlActionContext ctx, String seltab) throws Exception`

Table 4: Events TabSetControl

Arguments:

ctx	ControlActionContext. See Section 5.1.
seltab	Id of the selected tab as has been assigned within the JSP page at the time of declaration of the tabs. 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>
```

The TabserControl can contain additional control elements.

2.5 BreadCrumbControl

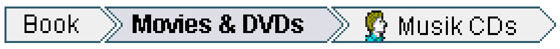


Figure 6: Example BreadCrumbControl

(Figure reduced in size)

The BreadCrumbControl generates the following events:

Trigger	Event	Signature of the Callback Method

BreadCrumb	Crumb click	*Ctrlname*_**`onCrumbClick`**`(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Table 5: Events BreadCrumbControl

Arguments:

ctx	`ControlActionContext. See Section 5.1.`
key	`Id of the selected crumb as has been assigned within the JSP page at the time of declaration of the crumbs. 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 Scheduler Control

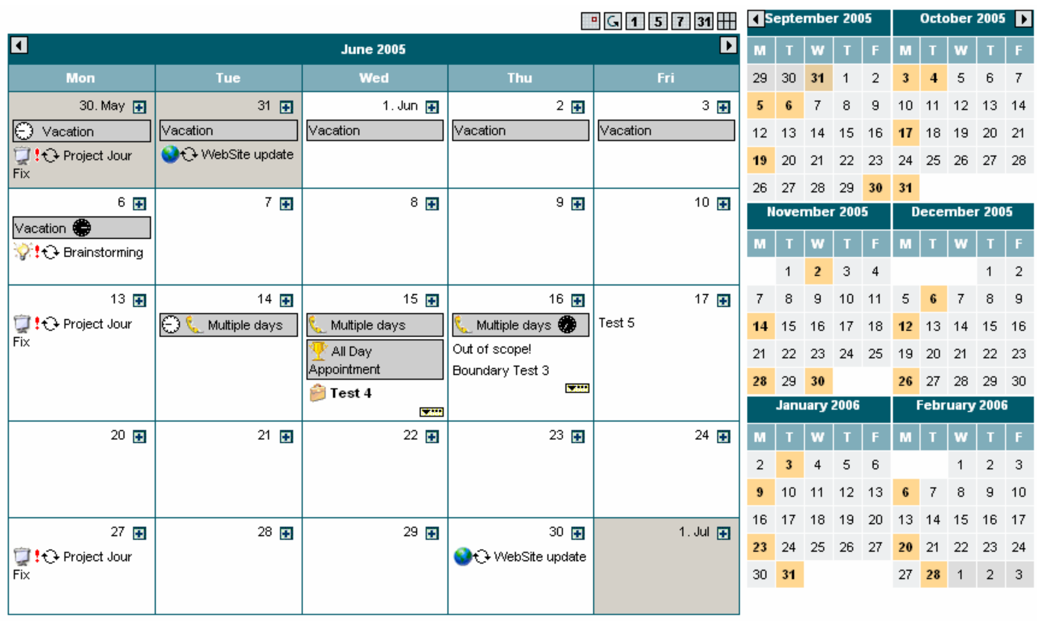


Figure 7: Example SchedulerControl

(Figure reduced in size)

The SchedulerCrumbControl generates the following events:

Trigger	Event	Signature of the Callback Method
Create-Button	Create	*Ctrlname*`_onCreate`(`ControlRequestContext ctx) throws Exception`
Add-Button	AddApointment	*Ctrlname*`_onAddAppointment`(`ControlRequestContext ctx,` `long timeInMillis) throws Exception`
Refresh-Button	Refresh	*Ctrlname*`_onRefresh`(`ControlRequestContext ctx) throws Exception`
Click on Appointment	Appointment Click	*Ctrlname*`_onAppointmentClick`(`ControlRequestContext ctx,` `String key,` `long timeInMillis) throws Exception`
Previous/ Next Date	ChangeDate	*Ctrlname*`_onChangeDate`(`ControlRequestContext ctx,` `long timeInMillis,` `String view) throws Exception`
Select Appointment	CheckAppointment	*Ctrlname*`_onCheckAppointment`(`ControlRequestContext ctx,` `String uniqueId,` `long timeInMillis,` `boolean check) throws Exception`
Select Day	CheckDate	*Ctrlname*`_onCheckDate`(`ControlRequestContext ctx,` `long timeInMillis,` `SchedulerScope scope,` `boolean check) throws Exception`

Drilldown Detail-View	SelectDate	*Ctrlname*`_onSelectDate(`
		`ControlRequestContext ctx,`
		`long timeInMillis,`
		`String view) throws Exception`

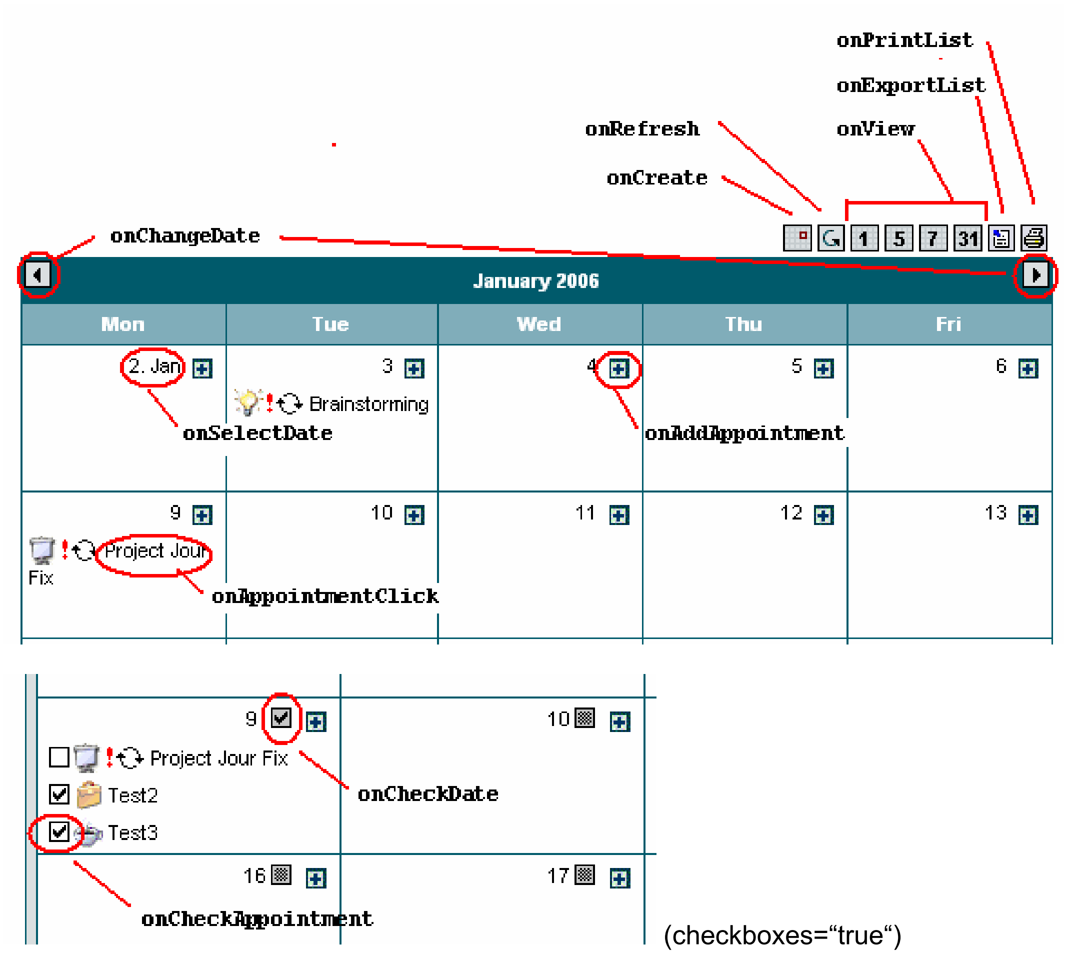
View-Button	View	*Ctrlname*`_onView(`
		`ControlRequestContext ctx,`
		`String view) throws Exception`

Print-Button	PrintList	*Ctrlname*`_onPrintList(` `ControlRequestContext ctx) throws Exception`
Export-Button	ExportList	*Ctrlname*`_onExportList(` `ControlRequestContext ctx) throws Exception`

Tabelle 6: Events SchedulerControl

Argumente:

ctx	ControlRequestContext.
timeInMillis	Date.
key	Id Appointments
view	Shows which view should be selected: day, workweek, week, month, year.
uniqueId	Id Appointments.
check	true if selected; false otherwise.



(checkboxes="true")

3 Callback Methods Form Buttons

Form buttons always generate an **onClick** event and in the Action class, result in a call to a corresponding Callback method. The only precondition for this is that the name of the button within the **name**-attribute should start with the prefix **"btn"**.

In addition, for using hover effects, the id-attribute for the bottom must be specified. In this case, the Id must also start with the prefix "**btn**". Also, the different images for the different states must be available.

The name of the Callback method is always composed of the name of the button (lowercase and without the "**btn"** prefix) and the suffix "**_onClick"**. The `FormActionContext` is always transferred to the method as the first parameter.

Form buttons generate the following event:

Trigger	Event	Signature of the Callback Method
Button	Click	*buttoname*_**onClick**(FormActionContext ctx)

Table 7: Event form element

The class `FormActionContextwird is described in chapter 0.`

Examples:

Nomenclature of the button in the JSP page:

```
<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-Tags:

```
<%@ 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 using the Struts and the Common-Controls Form-Tags:

```
<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>
```

Relevant Callback methods in the ActionClass:

For the button used in the JSP page, the following two Callback methods are included in the Action class:

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

```
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.
       * @param  ctx  FormActionContext
       * @throws Exception
       */
      public void save_onClick(FormActionContext ctx)  throws Exception {

            // save changes

      ctx.forwardByName(Forwards.SUCCESS);
      }
}
```

It is imperative to note the correct way of writing this, since otherwise, the method implemented cannot be found and invoked via the Reflection API, which can result in a Framework exception! The outcome of this is that the doExecute() method is called.

4 Functioning

The way in which control element events are forwarded from the Framework to the Eventhandler Callback methods depends on the configured "formelement" of the controlling. The differentiation between the two methods is transparent to the developer. Thus, there is no distinction in the implementation of the Event Handler for the application developer.

4.1 formelement="false"

All control element events trigger a hyperlink, which contains all the necessary parameters for forwarding to the Event Handler. If a control element so configured is used in an HTML form, then all the user inputs in the form are lost during the Server Roundtrip.
The generated control element hyperlink has the following elements:

Parameter	**Description**
ctrl	Name of the control element that has triggered the event.
action	Identifies the action triggered by the user (drilldown, edit, delete, etc...)
key	Key for identifying the element on which the action is to be carried out.

4.2 formelement="true"

No hyperlinks are generated. Rather, the form in which the control element is embedded is sent to the server with a control element action. Then, by means of automatically generated Hidden Fields, the suitable Event-Handler is called. The advantage here is that the user inputs in the HTML-form are not lost during the Server Roundtrip!

4.3 Event Dispatching

When there is an incoming request, the RequestProcessor of Struts determines the relevant action class that has to accept the request for processing. The request is then parsed by the higher-level class FWAktion, from which the called action is derived. In doing so, a check is carried out whether the request originates from a control element or a form element (see Figure 1).

In the case of a control element, the corresponding Eventhandler in the action class is called by means of the name of the control and the triggered event. The name of the control element is determined according to the **name of the Bean** which takes up the instance of the control element.

Similar to Struts, the following applies thereby:

- If the ListControl is passed directly, e.g. within the Session → then the bean name corresponds to the name of the attribute under which the bean has been stored in the session.
- If the ListControl is located within an ActionForm → then the bean name corresponds to the name of the Properties under which the instance of the control is stored within the form.

Example for 1:

In this example, the instance of the ListControl is saved within the session.

```
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();
    }
}
```

The Bean is accessed within the JSP page via the **name** attribute. In this example, the control element is not embedded in a form (in contrast to example 2), and therefore, the **action** attribute must be additionally specified, via which the events are forwarded to the Action class.

```
<ctrl:list
    action="sample101/userBrowse"
    name="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>
```

Example for 2:

The ListControl is located in an ActionForm. Since the ListControl must administer its own status across several ServerRoundtrips, in the struts-config.xml file, the form is stored in the scope “Session”. If the instance is held within the ActionForm, the data is initialized within the Action class. The method `setDataModel(ListDataModel dataModel)` of the ActionForm is used for the purpose, for example.

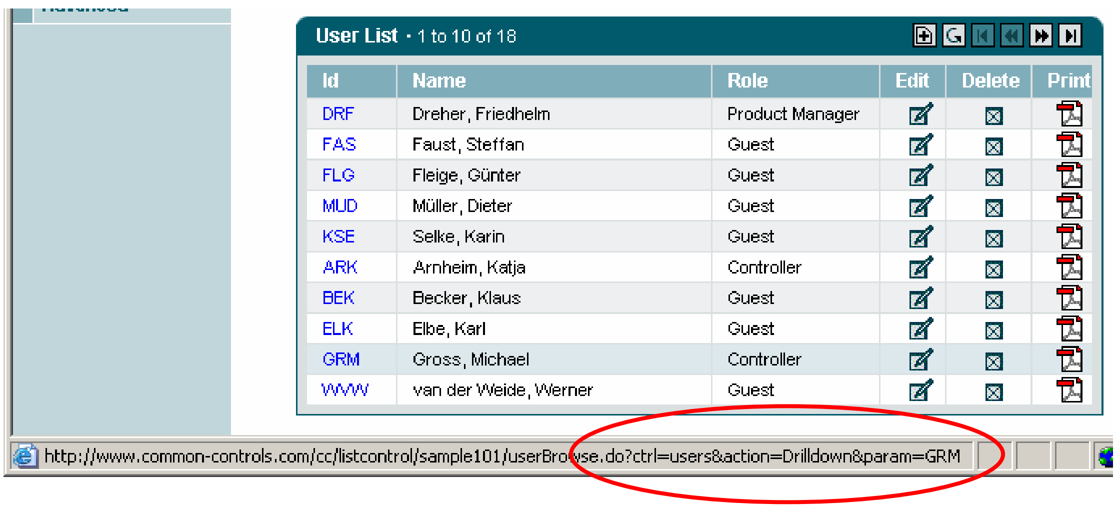
```
public class UserBrowseActionForm extends FWActionForm {

    /**
     * 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);
    }

}
```

The Bean is accessed within the JSP page via the **property** attribute. If the property Attribute is specified, then the control element automatically searches for its data model within the ActionForm that is assigned to the action. The instance is then accssed via the specified property, which in our example results in a call to the method `getUsers()`.

If the control element is embedded in a form, the **action** attribute is not specified any more.

```
<%@ 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 the property attribute is used, care should be taken that the getter and setter method accept or return the same object type.
The name of the Eventhandler that is triggered during an action on a control element can also be read in the status line of the browser. In the following example, the user clicks in the column “Id” on the data record “GRM” to go to the detail view.



User List · 1 to 10 of 18

Id	Name	Role	Edit	Delete	Print
DRF	Dreher, Friedhelm	Product Manager	✎	☒	📄
FAS	Faust, Steffan	Guest	✎	☒	📄
FLG	Fleige, Günter	Guest	✎	☒	📄
MUD	Müller, Dieter	Guest	✎	☒	📄
KSE	Selke, Karin	Guest	✎	☒	📄
ARK	Arnheim, Katja	Controller	✎	☒	📄
BEK	Becker, Klaus	Guest	✎	☒	📄
ELK	Elbe, Karl	Guest	✎	☒	📄
GRM	Gross, Michael	Controller	✎	☒	📄
WVW	van der Weide, Werner	Guest	✎	☒	📄

```
http://www.common-controls.com/cc/listcontrol/sample101/userBrowse.do?ctrl=users&action=Drilldown&param=GRM
```

The hyperlink generated thereby has the following parameters:

Parameter	Meaning	Value
ctrl	Name of the control element that has triggered the event.	users
action	Identifies the action initiated by the user (drilldown, edit, delete, ...)	Drilldown
key	Key for identifying the element on which another action is to be executed.	GRM

The Callback method in our Action class has the name `users_onDrilldown` and along with the ControlActionContect, is passed the key “GRM”.

5 Classes

5.1 ControlActionContext

The interface `ControlActionContext` extends the interface `ActionContext` with methods that are required for the handling of events of control elements.

The following actions can be carried out via the Interface within a Callback method:

Methode	**Description**
mapping()	Is used for accessing the ActionMapping object.
form()	Can be used for accessing the FormBean which is assigned to the Action in the struts-config.xml file.
request()	Serves to access the Request object.
response()	Serves to access the Response object.
session()	Serves to access the Session object.
forwardToInput()	Forward to the page specified in the ActionMapping, which is specified by means of the input Attribute.
forwardToAction()	Forwarded to the specified Action.
forwardByName()	Searches for an ActionForward via the specified name
getErrors()	Returns the error collection
getMessages()	Returns the message collection
addError()	Serves to include an exception in the global error collection. **Remark:** The Framework, as against Struts, rescues the error and message text collection across several redirects till the next JSP page is called.
addGlobalError()	Saves a general error (without reference to an input field) and is used to present error messages upon returning from a sub-dialog within the calling dialog.
addPropertyError()	Saves an error to a Property in the error collection
addMessage()	Serves to include a message in the global message collection
addGlobalMessage()	Saves a message without reference to a Property. Is used to present messages upon returning from a sub-dialog within the calling dialog.
addPropertyMessage()	Saves a message for a Property in the Message collection
hasMessages()	Checks whether there are messages present.
hasErrors()	Checks whether there are error messages.
getPrincipal()	Returns the Principal Object (cc-framework Security System)
control()	Returns the instance of the control element that has triggered the event
action()	Returns the metadata of this control action (for instance “Drilldown”)
getActionMethod()	Returns the name of the method that is invoked by the Actionhandler upon occurrence of the event. In the form [Controlname]_on[Action].

5.2 FormActionContext

The interface `ControlActionContext` extends the Interface `ActionContext` with methods that are required for the handling of events from buttons.

Methode	Description
getActionMethod()	Returns the name of the method that is called after the occurrence of the event by the Actionhandler. In the form `[formularname]_on[Action].`
Additional methods are described in Section 5.1.	

5.3 SelectMode

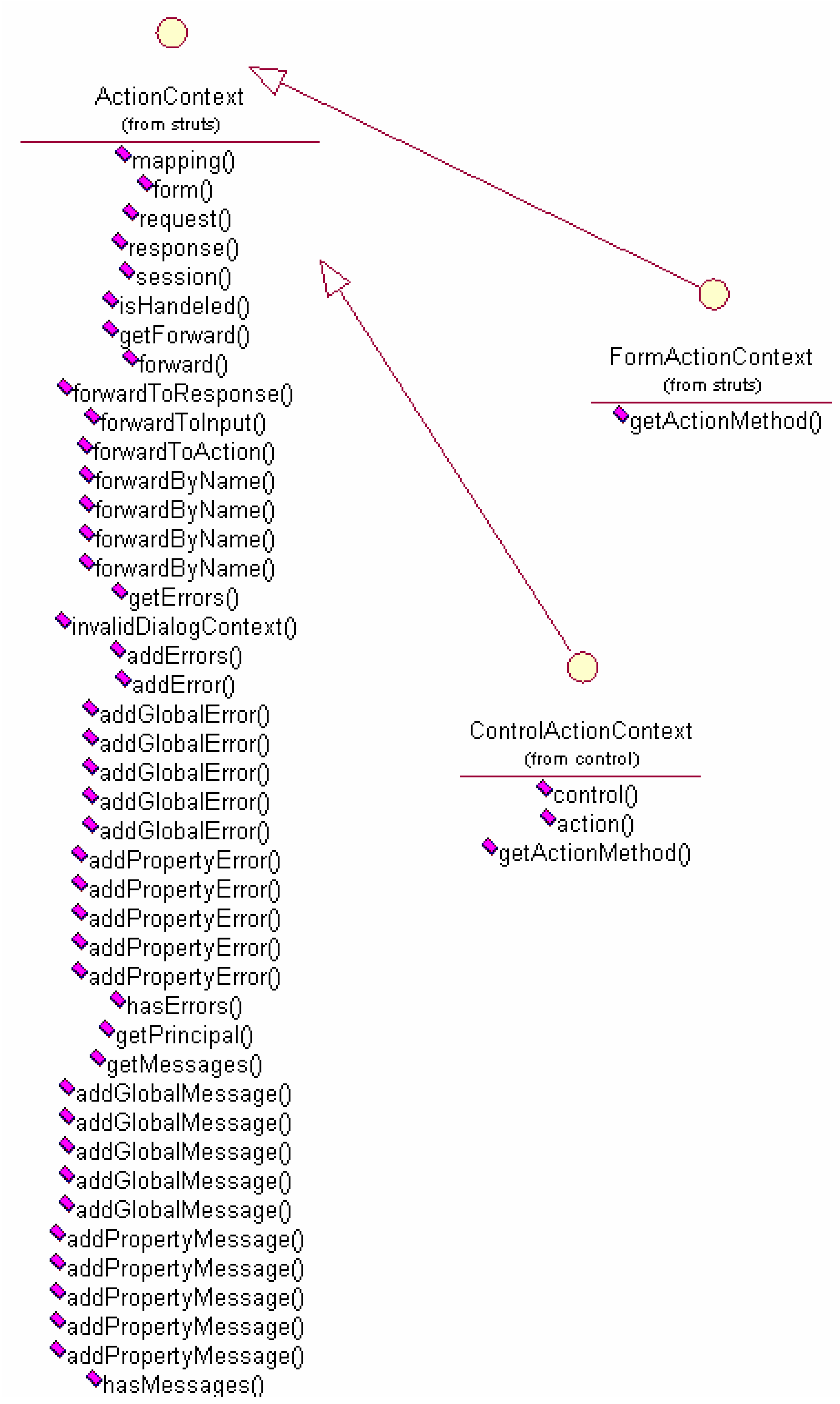
Value	Description
`NONE`	No selection mode specified.
`SINGLE`	Only one element may be selected.
`MULTIPLE`	Any number of elements may be selected.

5.4 SortOrder

Value	Description
`NONE`	No sorting order specified.
`ASCENDING`	Sorting in ascending order.
`DESCENDING`	Sorting in descending order.

6 Appendix

6.1 Classdiagram ActionContext



7 Support

We would be happy to be of service if you have any questions or problems. Please use our Service Form on our homepage for your queries. We shall endeavor to answer your queries as quickly as possible