Common-Controls Events & Callback-Methoden

Version 1.6.0 - Stand: 14. Januar 2006

Common Controls

Herausgeber:
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

Inhaltsverzeichnis

1 Übersicht

Dieses Dokument beschreibt, wie innerhalb des Common Controls Frameworks Callback-Methoden zur Behandlung von User Interface Ereignissen implementiert werden.

Das Framework unterscheidet prinzipiell zwischen Callback Methoden für ***Kontrollelemente*** und ***Formularelemente***. Für beide Arten können Callback-Methoden innerhalb einer Aktion Klasse implementiert werden.

Bei einem Event von einem Kontrollelemente bekommt die Callback Methode neben weiteren Parametern immer den `ControlActionContext` übergeben, während Callback Methoden von Formularen beim Aufruf den `FormActionContext` erhalten.

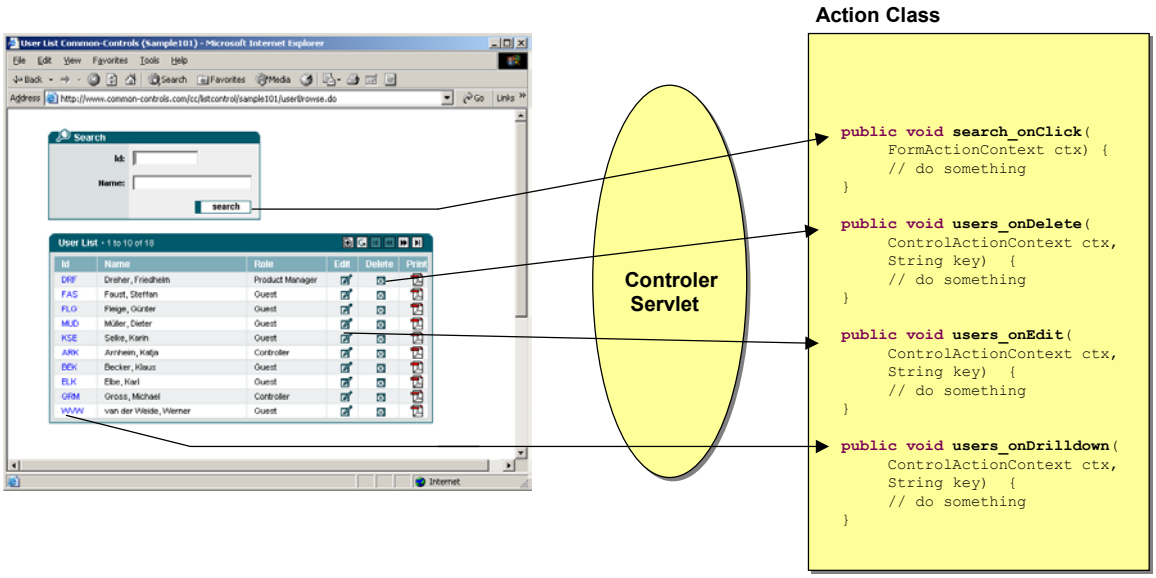


Abbildung 1: Callback Methoden für Kontroll- und Formelemente

Bei dem `ControlActionContext und dem FormActionContext` handelt es sich jeweils um Interfaces, die im wesentlichen den Zugriff auf Objekte wie das Session-, Request- oder Response- Objekt kapseln. Darüber hinaus liefern sie zusätzliche Informationen für das eingetretene Ereignis, wie zum Beispiel eine Referenz auf das Kontrollelement welches das Ereignis ausgelöst hat. Beide Interfaces werden im Abschnitt 5 näher beschrieben.

2 Callback Methoden Controls

2.1 ListControl

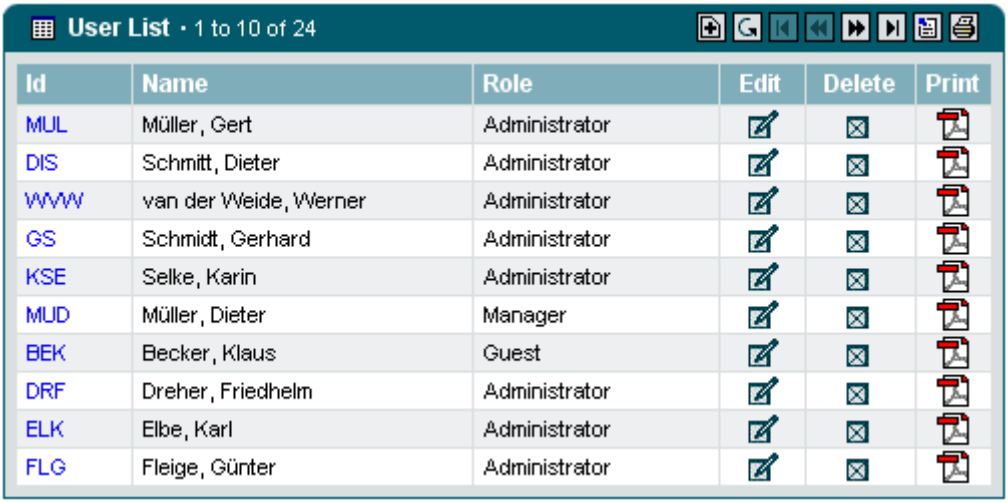


Abbildung 2: Beispieldarstellung ListControl

Das ListControl generiert die folgenden Events:

Auslöser	Event	Signatur der Callback Methode

Drilldown Spalte	Drilldown	*Ctrlname*`_onDrilldown(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Edit Spalte	Edit	*Ctrlname*`_onEdit(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Delete Spalte	Delete	*Ctrlname*`_onDelete(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Selector Spalte	Select	*Ctrlname*`_onSelect(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Button Spalte	CellClick	*Ctrlname*`_onCellClick(`
		`ControlActionContext ctx,`
		`String column,`
		`String key) throws Exception`
		`Bei Angabe des „command” Attributes wird die entsprechende Methode aufgerufen`
		Ctrlname`_on`*Command*`(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Image Spalte	CellClick	Ctrlname`_onCellClick(`
		`ControlActionContext ctx,`
		`String column,`
		`String key) throws Exception`

Check Spalte	Check	*CtrlName*_**onCheck**(ControlActionContext ctx, String key, SelectMode mode, boolean checked) throws Exception
Checkbox Spalte	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
Bei Seitenwechsel	Page	*CtrlName*_**onPage(** ControlActionContext ctx, int page) throws Exception

Tabelle 1: Events ListControl

Argumente:

ctx	ControlActionContext. Siehe Abschnitt 5.1.
column	Name der Spalte, in der das Ereignis ausgelöst wurde. Als Name wird das Property der Spalte zurückgeliefert.
direction	SortOrder. Die Sortierrichtung innerhalb der Spalte (Aufsteigend oder Absteigend)
key	Eindeutiger Schlüssel der Zeile, wie er vom ListDataModel geliefert wurde.
mode	SelectMode. Gibt an, ob in einer Checkboxspalte nur ein Element auswählbar ist oder gleichzeitig mehrere selektiert werden dürfen.
page	Die Seite auf die gesprungen werden soll. -1 = Auf die letzte Seite springen 0..n = Auf die angegebene Seite springen Achtung: Der Handler muss selbst auf die Einhaltung des gültigen Bereiches achten. Diese Methode sollte nur dann implementiert werden, wenn ein anderes Blätterverhalten in einer Liste benötigt wird.

2.2 TreeControl

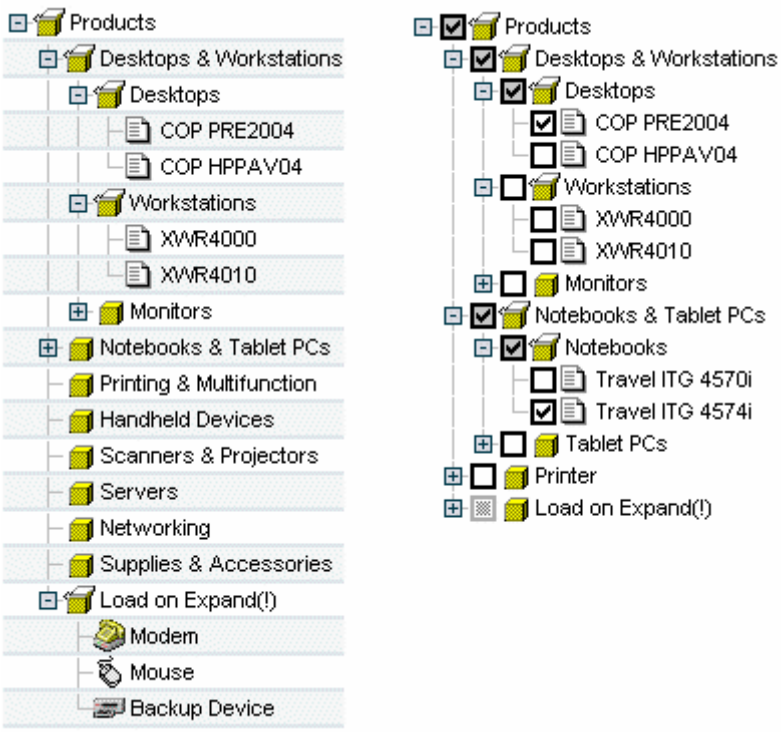


Abbildung 3: Beispieldarstellung TreeControl

Das TreeControl generiert die folgenden Events:

Auslöser	Event	Signatur der Callback Methode
Öffnen Gruppenknoten	Expand	*CtrlName*`_onExpand(` `ControlActionContext ctx,` `String key)` *`throws Exception`*

Öffnen Gruppenknoten mit unbekanter Anzahl Kinder	ExpandEx	*CtrlName*`_onExpandEx(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Schließen Gruppenknoten	Collapse	*CtrlName*`_onCollapse(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Klick auf Checkbox	Check	*CtrlName*`_onCheck(`
		`ControlActionContext ctx,`
		`String key,`
		`SelectMode mode,`
		`boolean checked) throws Exception`

Tabelle 2: Events TreeControl

Argumente:

`ctx`	`ControlActionContext. Siehe Abschnitt 5.1.`
`key`	`Eindeutiger Schlüssel der Zeile, wie er von dem TreeNodeDataModel geliefert wurde.`
`mode`	`SelectMode. Gibt an, ob in einer Checkboxspalte nur ein Element auswählbar ist oder gleichzeitig mehrere selektiert werden dürfen.`

2.3 TreeListControl

Abbildung 4: Beispieldarstellung TreeListControl

(Abbildung verkleinert)

Das TreeListControl kombiniert einen Baum mit einer Liste. Folglich kann es *alle* Events die im Kapitel über das TreeControl und das ListControl beschrieben sind generieren. An dieser Stelle soll nur auf die Events eingegangen werden, die zusätzlich auftreten können.

Das TreeListControl generiert die folgenden zusätzlichen Events:

Auslöser	Event	Signatur der Callback Methode

Add Spalte	Add	*CtrlName*`_onAdd(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Tabelle 3: Events TreeListControl

ctx `ControlActionContext. Siehe Abschnitt 5.1.`

key `Eindeutiger Schlüssel der Zeile, wie er von dem TreeNodeDataModel geliefert wurde.`

2.4 TabSetControl

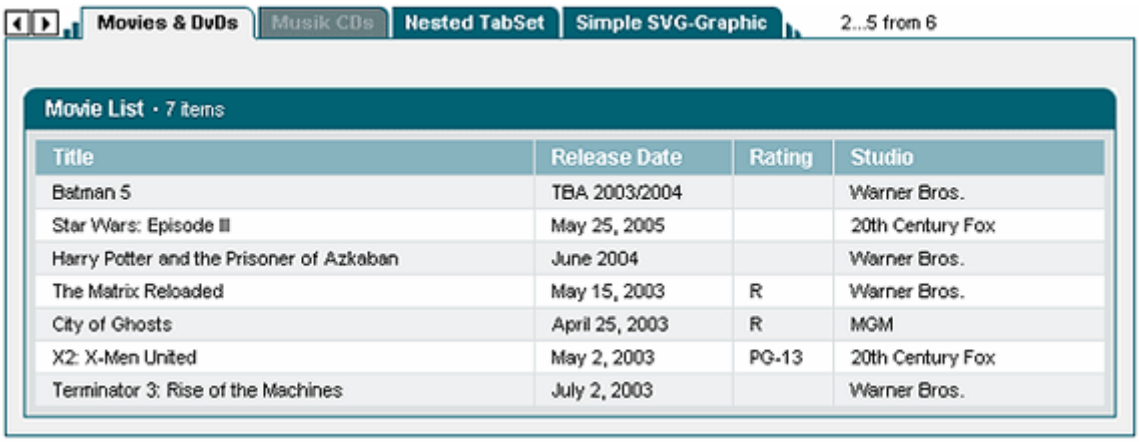


Abbildung 5: Beispieldarstellung TabSetControl

(Abbildung verkleinert)

Das TabSetControl generiert die folgenden Events:

Auslöser	Event	Signatur der Callback Methode

TabPage	TabClick	*Ctrlname*`_onTabClick(`
		`ControlActionContext ctx,`
		`String seltab) throws Exception`

Tabelle 4: Events TabSetControl

Argumente:

ctx ControlActionContext. Siehe Abschnitt 5.1.
seltab Id der selektierten Tabe (Reiter), wie sie innerhalb der JSP-Seite bei der Deklaration der Tabe vergeben wurde. Im Beispiel unten entweder tab1, tab2 oder 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>
```

Ein TabSetControl kann weitere Kontrollelemente enthalten.

2.5 BreadCrumbControl

Abbildung 6: Beispieldarstellung BreadCrumbControl

(Abbildung verkleinert)

Das BreadCrumbControl generiert die folgenden Events:

Auslöser	Event	Signatur der Callback Methode

BreadCrumb	Crumb click	*CtrlName*`_onCrumbClick(`
		`ControlActionContext ctx,`
		`String key) throws Exception`

Tabelle 5: Events BreadCrumbControl

Argumente:

ctx ControlActionContext. Siehe Abschnitt 5.1.
key Id des selektierten Crumbs, wie sie innerhalb der JSP-Seite bei der Deklaration des BreadCrumbs vergeben wurde. Im Beispiel unten entweder crumb1, crumb2 oder 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

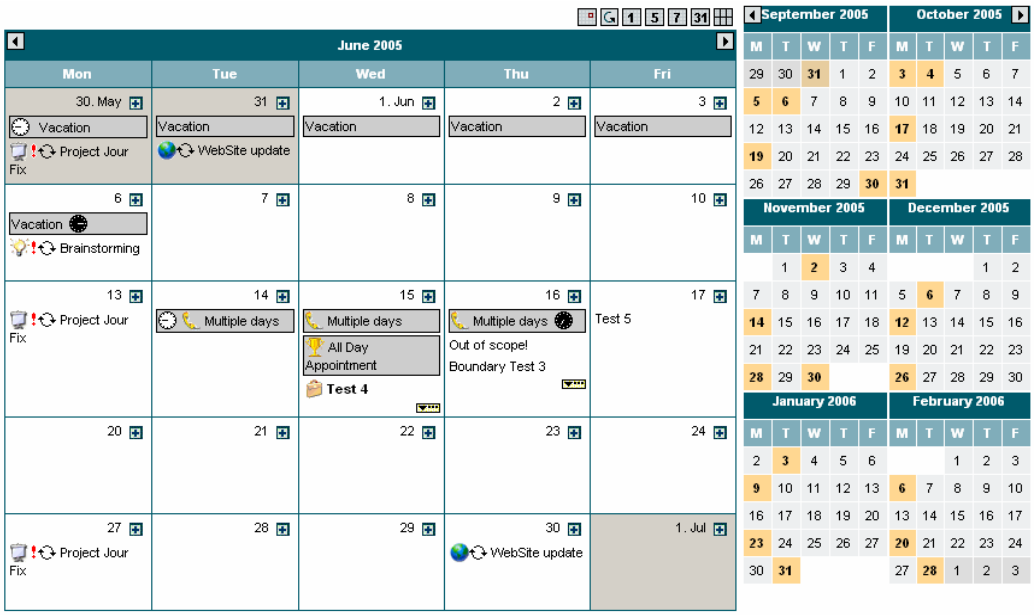


Abbildung 7: Beispieldarstellung SchedulerControl

(Abbildung verkleinert)

Das SchedulerCrumbControl generiert die folgenden Events:

Auslöser	Event	Signatur der Callback Methode
Create-Button	Create	*CtrlName*`_onCreate(` `ControlRequestContext ctx) throws Exception`

Add-Button zum hinzufügen mit Tagesbezug	AddApointment	*CtrlName*`_onAddAppointment(`
		`ControlRequestContext ctx,`
		`long timeInMillis)`

Refresh-Button	Refresh	*CtrlName*`_onRefresh(` `ControlRequestContext ctx) throws Exception`

Click auf Appointment	AppointmentClick	*CtrlName*`_onAppointmentClick(`
		`ControlRequestContext ctx,`
		`String key,`
		`long timeInMillis)`

Vorhergehendes/ Nächstes Datum	ChangeDate	*CtrlName*`_onChangeDate(`
		`ControlRequestContext ctx,`
		`long timeInMillis,`
		`String view)`

Auswahl Appointment	CheckAppointment	*CtrlName*`_onCheckAppointment(`
		`ControlRequestContext ctx,`
		`String uniqueId,`
		`long timeInMillis,`
		`boolean check)`

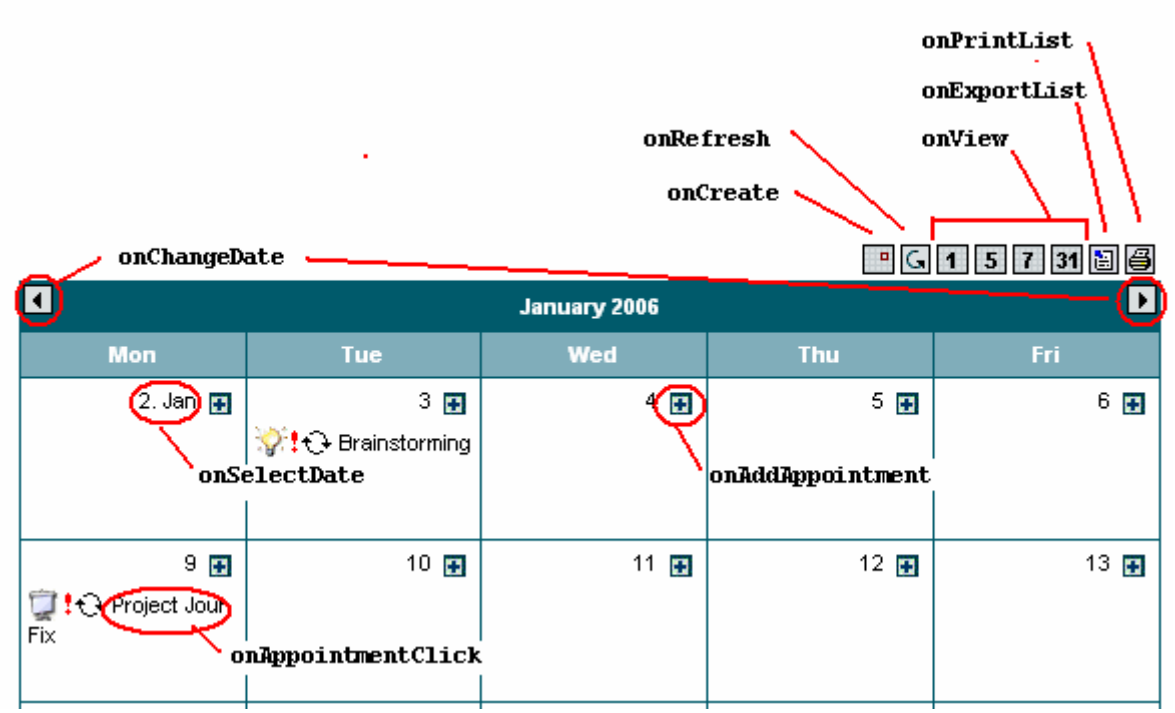
Auswahl Tag	CheckDate	*CtrlName*`_onCheckDate(`
		`ControlRequestContext ctx,`
		`long timeInMillis,`
		`SchedulerScope scope,`
		`boolean check)`

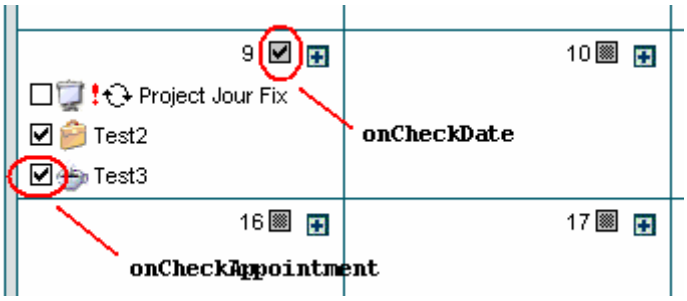
Verzweigung Detailansicht	SelectDate	*CtrlName*_**onSelectDate**(ControlRequestContext ctx, long timeInMillis, String view)
View-Button	View	*CtrlName*_**onView**(ControlRequestContext ctx, String view)
PrintButton	PrintList	*CtrlName*_**onPrintList**(ControlRequestContext ctx) throws Exception
ExportButton	ExportList	*CtrlName*_**onExportList**(ControlRequestContext ctx) throws Exception

Tabelle 6: Events SchedulerControl

Argumente:

ctx	ControlRequestContext.
timeInMillis	Datum
key	Id Appointments
view	Zeigt an auf welche Sicht umgeschaltet werden soll: day, workweek, week, month, year
uniqueId	Id Appointments
check	true wenn ausgewählt; false sonst





(checkboxes="true")

3 Callback Methoden Formularschaltflächen

Formularschaltflächen generieren immer ein **onClick** Event und führen in der Action Klasse zu einem Aufruf einer entsprechenden Callback-Methode. Die Einzige Voraussetzung hierfür ist, dass der Name des Buttons innerhalb des **name**-Attributes mit dem Präfix „**btn“** beginnt.

Für die Nutzung von Hover Effekten muss zudem das id-Attribute für den Button angegeben werden. Die Id muss in diesem Fall ebenfalls mit dem Präfix „**btn“** beginnen. Zudem müssen die unterschiedlichen Images für die verschiedenen Zustände vorliegen.

Der Name der Callback-Methode setzt sich immer aus dem Namen des Buttons (kleingeschrieben und ohne das „**btn“** Präfix) und dem Suffix „**_onClick“** zusammen. Der Methode wird als erster Parameter immer der `FormActionContext` übergeben.

Formularschaltflächen generieren das folgenden Event:

Auslöser	Event	Signatur der Callback Methode
Button	Click	*buttoname*`_onClick(` `FormActionContext ctx)`

Tabelle 7: Event Formularelement

Die Klasse `FormActionContext`wird im Kapitel 5.2 beschrieben.

Beispiele:

Benennung des Buttons in der JSP Seite:

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

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

oder bei Verwendung des Struts Form-Tags:

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

<html:form action="usereedit.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>
```

oder bei Verwendung des Struts- und der 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>
```

Zugehörige Callback Methoden in der ActionKlasse:

Für die in der JSP Seite verwendeten Button werden in der Action Klasse die beiden folgenden Callback Methoden aufgenommen:

- `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);
    }
}
```

Es muss unbedingt auf die richtige Schreibweise geachtet werden, da sonst die Implementierte Methode nicht über die Reflection API gefunden und aufgerufen werden kann, was zu einer Framework Exception führt! Dies führt dann dazu, dass die doExecute() Methode aufgerufen wird.

4 Funktionsweise

Die Art, wie Kontrollelement Ereignisse vom Framework an die Eventhandler Callback Methoden weitergeleitet werden, hängt von der konfigurierten Eigenschaft „formelement“ des Kontrollelementes ab. Die Unterscheidung zwischen den beiden Methoden erfolgt für den Entwickler transparent. Es gibt also keine Unterscheidung in der Implementierung der Event Handler für den Anwendungsentwickler.

4.1 formelement=“false“

Alle Kontrollelement Ereignisse lösen einen Hyperlink aus, welcher alle notwendigen Parameter zur Weiterleitung an den Event Handler enthält. Wird ein so konfiguriertes Kontrollelement in einem HTML-Formular verwendet, dann gehen beim Server Roundtrip alle Benutzereingaben in dem Formular verloren!
Der generierte Kontrollelement Hyperlink besitzt die folgenden Elemente:

Parameter	Bedeutung
ctrl	Name des Kontrollelementes, das das Event ausgelöst hat.
action	Identifiziert die von dem Benutzer ausgelöste Aktion (drilldown, edit, delete, etc...)
key	Schlüssel zur Identifikation des Elementes auf dem die Aktion ausgeführt werden soll.

4.2 formelement=“true“

Es werden keine Hyperlinks generiert. Vielmehr wird das Formular in welchem das Kontrollelement eingebettet ist bei einer Kontrollelement Aktion an den Server abgeschickt. Mit Hilfe von automatisch generierten Hidden Fields erfolgt dann der Aufruf des passenden Event-Handlers. Der Vorteil hierbei ist, dass die Benutzereingaben im HTML-Formular bei dem Server Roundtrip nicht verloren gehen!

4.3 Event Dispatching

Der RequestProcessor von Struts ermittelt bei einem eingehenden Request die zugehörige Aktionklasse, welche den Request zur Bearbeitung entgegen nehmen soll. Der Request wird dann von der Oberklasse FWAktion, von der die aufgerufene Aktion abgeleitet ist, geparst. Dabei wird überprüft, ob der Request von einem Kontrollelement oder Formelement stammt (vgl. Abbildung 1).

Bei einem Kontrollelement wird anhand des Namens des Controls und dem ausgelösten Event der entsprechende Eventhandler in der Action Klasse aufgerufen. Der Name des Kontrollelementes bestimmt sich nach dem **Namen der Bean**, welche die Instanz des Kontrollelementes aufnimmt.

Dabei gilt analog zu Struts:

- Wird das ListControl direkt übergeben, z.B. innerhalb der Session --> Dann entspricht der Beanname dem Namen des Attributes, unter dem die Bean in der Session abgelegt wurde.
- Befindet sich das ListControl innerhalb eines ActionForms --> Dann entspricht der Beanname dem Namen des Properties unter dem die Instanz des Controls innerhalb des Forms abgelegt ist.

Beispiel zu 1:

In diesem Beispiel wird die Instanz des ListControls innerhalb der Session abgelegt.

```
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();
     }
}
```

Der Zugriff auf die Bean erfolgt innerhalb der JSP Seite über das **name** Attribute. In diesem Beispiel ist das Kontrollelement nicht in einem Formular eingebetet (im Gegensatz zu Beispiel 2), daher muss zusätzlich das **action** Attribut angegeben werden über das die Ereignisse an die Aktion Klasse weitergeleitet werden.

```
<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>
```

Beispiel zu 2:

Das ListControl befindet sich in einem ActionForm. Da das ListControl seinen Zustand über mehrere Serverroundtrips selbst verwalten muss, wird in der struts-config.xml Datei das Formular im Scope „Session“ abgelegt. Wenn die Instanz innerhalb des ActionForms gehalten wird, erfolgt die Initialisierung der Daten innerhalb der Action Klasse. Dazu wird etwa die Methode `setDataModel(ListDataModel dataModel)` des ActionForms benutzt.

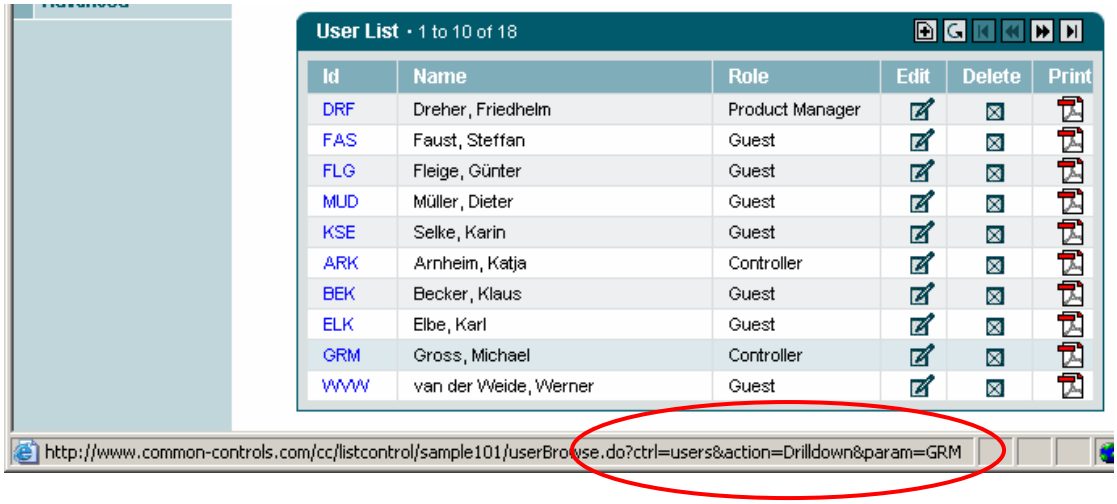
```
public class UserBrowseActionForm extends FWActionForm {

    /**
     * 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);
    }

}
```

Der Zugriff auf die Bean erfolgt innerhalb der JSP Seite über das **property** Attribute. Wird das Property Attribut angegeben, dann sucht das Kontrollelement sein Datenmodell automatisch innerhalb des ActionForms, welches der Action zugeordnet ist. Der Zugriff auf die Instanz erfolgt dabei über das spezifizierte Property welches in unserem Beispiel zum Aufruf der Methode `getUsers()` führt.
Wenn das Kontrollelement in ein Formular eingebetet ist, wird das **action** Attribut nicht mehr angegeben.

```
<%@ 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/>
```

Bei der Verwendung des `property` Attributes ist immer darauf zu achten, dass die getter und setter Methode den gleichen Objekttyp entgegennehmen bzw. zurückliefern!

Der Name des Eventhandlers, welcher bei einer Aktion auf einem Kontrollelement ausgelöst wird, kann auch in der Statuszeile des Browsers abgelesen werden. In dem folgenden Beispiel klickt der Anwender in der Spalte „Id“ auf den Datensatz „GRM“ um in die Detailansicht zu gelangen.



Der dabei generierte Hyperlink hat die folgenden Parameter:

Parameter	Bedeutung	Wert
ctrl	Name des Kontrollelementes, das das Event ausgelöst hat.	users
action	Identifiziert die von dem Benutzer ausgelöste Aktion (drilldown, edit, delete, etc...)	Drilldown
key	Schlüssel zur Identifikation des Elementes auf dem eine weitere Aktion ausgeführt werden soll.	GRM

Die Callback-Methode in unsere Aktion Klasse trägt als den Namen users_onDrilldown und bekommt neben dem ControlActionContect den Schlüssel „GRM“ übergeben.

5 Klassen

5.1 ControlActionContext

Das Interface `ControlActionContext` erweitert das Interface `ActionContext` um Methoden, die für die Behandlung von Events von Kontrollelementen notwendig sind.

Über das Interface können innerhalb einer Callback Methode folgende Aktionen durchgeführt werden:

Methode	Beschreibung
mapping()	Dient dem Zugriff auf das ActionMapping Objekt.
form()	Dient dem Zugriff auf die FormBean, welche der Action in der struts-config.xml Datei zugeordnet ist
request()	Dient dem Zugriff auf das Request Objekt.
response()	Dient dem Zugriff auf das Response Objekt.
session()	Dient dem Zugriff auf das Session Objekt.
forwardToInput()	Forwarded zu der im ActionMapping angegebenen Seite, welche über das input Attribut spezifiziert ist.
forwardToAction()	Forwarded zu der angegebenen Action.
forwardByName()	Sucht ein ActionForward über den angegebenen Namen
getErrors()	Liefert die Fehlerkollektion
getMessages()	Liefert die Messagekollektion
addError()	Dient zur Aufname einer Exception in die globale Fehlerkollektion **Anmerkung:** Das Framework rettet im Gegensatz zu Struts die Fehler- und Meldungstext Kollektion über mehrere redirects hinweg bis zum Aufruf der nächsten JSP-Seite!
addGlobalError()	Speichert einen allgemeinen Fehler (ohne Bezug zu einem Eingabefeld) und dient dazu Fehlermeldungen bei der Rückkehr aus einem Subdialog innerhalb des aufrufenden Dialogs zu präsentieren.
addPropertyError()	Speichert einen Fehler zu einem Property in der Fehlerkollektion
addMessage()	Dient zur Aufname einer Meldung in die globale Meldungskollektion
addGlobalMessage()	Speichert eine Meldung ohne Bezug zu einem Property. Dient dazu Meldungen bei der Rückkehr aus einem Subdialog innerhalb des aufrufenden Dialogs zu präsentieren.
addPropertyMessage()	Speichert eine Meldung zu einem Property in der Message Kollektion
hasMessages()	Prüft, ob Meldungen vorhanden sind.
hasErrors()	Prüft, ob Fehlermeldungen vorhanden sind.
getPrincipal()	Liefert das Principal Object (cc-framework Security System)
control()	Liefert die Instanz des Kontrollelementes, welches das Event ausgelöst hat
action()	
getActionMethod()	Liefert den Namen der Methode, welche nach Auftreten des Events durch den Actionhandler aufgerufen wird. In der Form `[Controlname]_on[Action].`

5.2 FormActionContext

Das Interface `ControlActionContext` erweitert das Interface `ActionContext` um Methoden, die für die Behandlung von Events von Buttons notwendig sind.

Methode	Beschreibung
getActionMethod()	Liefert den Namen der Methode, welche nach Auftreten des Events durch den Actionhandler aufgerufen wird. In der Form `[formularname]_on[Action].`
Die weiteren Methoden sind in Abschnitt 5.1 beschrieben.	

5.3 SelectMode

Wert	Bedeutung
`NONE`	Keine Selectionsmodus angegeben.
`SINGLE`	Nur ein Element darf selektiert werden.
`MULTIPLE`	Beliebig viele Elemente dürfen selektiert werden.

5.4 SortOrder

Wert	Bedeutung
`NONE`	Keine Sortierreihenfolge angegeben.
`ASCENDING`	Sortierung Aufsteigend.
`DESCENDING`	Sortierung Absteigend.

6 Anhang

6.1 Klassendiagramm ActionContext