SCC
INFORMATIONSSYSTEME

COMMON CONTROLS · GUIDED TOUR

ListControl

Das Kontrollelement Schritt für Schritt in eine bestehende Anwendung einbauen

SCC Informationssysteme GmbH

Version 1.6 · Stand: 14.01.2006

Impressum

HERAUSGEBER

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

Inhalt

1 Guided Tour ListControl

1.1 Gegenstand

Diese Übung demonstriert den Einsatz des ListControls. Das Kontrollelement erzeugt eine Tabelle, deren Aufbau und Aussehen frei konfigurierbar ist. Der Blättermechanismus, die Sortierung innerhalb der Spalten oder die Aktualisierung des Datenmodels beim Klick auf die Checkspalte müssen nicht implementiert werden. Diese Grundfunktionen deckt das ListControl bereits ab.

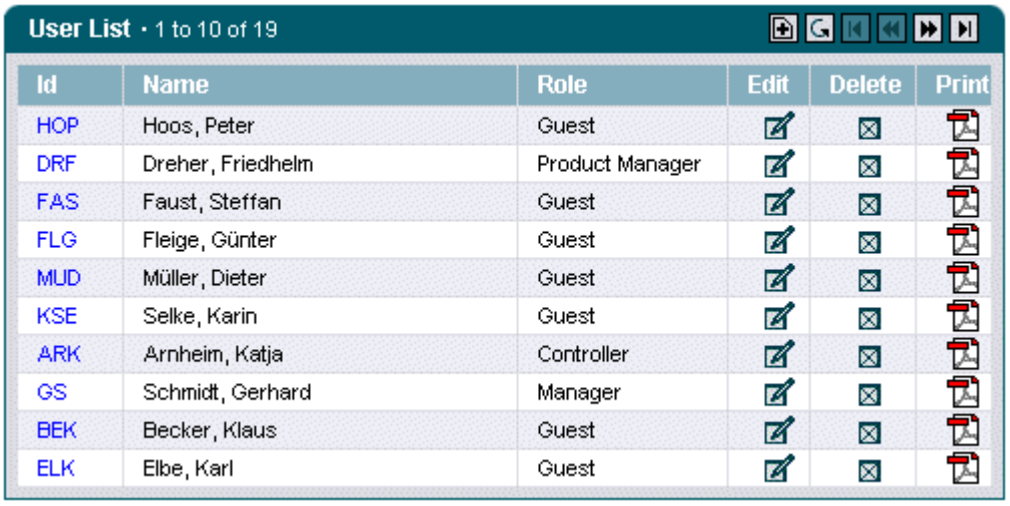


Abbildung 1: Beispieldarstellung ListControl

Zum Einsatz des ListControls sind die folgenden Schritte notwendig:

1. Auswahl des Designs für die Benutzeroberfläche
2. Erstellung einer Action-Klasse
3. Instanziierung eines ListControls
4. Bereitstellung der Anzeigedaten
5. Konfiguration der Tabelle innerhalb der JSP-Seite

1.2 Registrierung der Painterfactory

Zuerst erfolgt die Registrierung der Painterfactory. Sie legt fest, welches Design die Benutzeroberfläche erhält. Dies kann in der init()-Methode des Frontcontroler-Servlets applikationsweit geschehen.[1] Wir wählen hier das Standarddesign, welches uns der DefaultPainter bereitstellt.[2]

JAVA

```java
import javax.servlet.ServletException

import org.apache.struts.action.ActionServlet;
import com.cc.framework.ui.painter.PainterFactory
import com.cc.framework.ui.painter.def.DefPainterFactory;
import com.cc.framework.ui.painter.html.HtmlPainterFactory;

public class MyFrontController extends ActionServlet {

    public void init() throws ServletException {

        super.init();

        // Register all Painter Factories with the preferred GUI-Design
        // In this case we use the Default-Design.
        PainterFactory.registerApplicationPainter (
            getServletContext (), DefPainterFactory.instance());
        PainterFactory.registerApplicationPainter (
            getServletContext(), HtmlPainterFactory.instance());
    }
}
```

[1] Wenn der einzelne Benutzer zwischen verschiedenen Oberflächendesigns wählen können soll, dann werden zusätzliche PainterFactorys in der Benutzersession registriert. Dies erfolgt meist in der LoginAction mit PainterFactory.registerSessionPainter() im Session Scope.

[2] Weitere Designs (PainterFactories) sind im Lieferumfang der Professional Edition enthalten, oder können selbst entwickelt werden.

1.3 Ableitung der Action Klasse für den Struts Adapter

Unsere Tabelle soll Informationen über die Systembenutzer anzeigen. Daher soll die Action-Klasse, welche das Laden und Befüllen des ListControls übernimmt, die Bezeichnung "UserBrowseAction" tragen. Die Actionklasse wird dabei von der Klasse FWAction abgeleitet, welche die Struts-Action Klasse kapselt und um Funktionalitäten des Präsentationsframeworks erweitert. Dabei wird anstelle der execute()-Methode die doExecute()-Methode aufgerufen. Sie erhält beim Aufruf den ActionContext, über den der Zugriff auf weitere Objekte, wie das Request-, Session- und Response-Objekt gekapselt ist.

JAVA

```java
import java.io.IOException;
import javax.servlet.ServletException;

import com.cc.framework.adapter.struts.FWAction
import com.cc.framework.adapter.struts.ActionContext

public class UserBrowseAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx)
        throws IOException, ServletException {
        // In the next chapter, we will instantiate
        // our ListControls with the DisplayData
    }
}
```

1.4 Instanziierung des ListControls

Nun wird das ListControl innerhalb der UserBrowseAction instanziiert und mit den anzuzeigenden Daten gefüllt. Dazu wird über die setDataModel()-Methode dem Kontrollelement sein Datenmodell zugeordnet. Die Methode setDataModel() nimmt als Argument dabei ein **ListDataModel** entgegen. Hierbei handelt es sich um ein einfaches Interface, das von der Klasse UserDisplayList, welche die Anzeigedaten bereitstellt, implementiert wird.

JAVA

```java
import java.io.IOException;
import javax.servlet.ServletException;

import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.ui.control.SimpleListControl;

import com.cc.sampleapp.common.Messages;
import com.cc.sampleapp.presentation.dsp.UserDisplayList;

public class UserBrowseAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx)
        throws IOException, ServletException {

        try {
            // Get the Displaydata for our List
            UserDisplayList dspData = DBUser.fetch();

            // Create the ListControl and populate it.
            // with the Data to be displayed
            SimpleListControl userList = new SimpleListControl();
            userList.setDataModel(dspData);

            // Put the ListControl into the Session-Object.
            // Our ListControl is a statefull Object.
            ctx.session().setAttribute("users", userList);
        }
        catch (Throwable t) {
            ctx. AddGlobalError(Messages.ERROR, t);
        }

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

1.5 Bereitstellen der Anzeigedaten

Die Klasse, welche die Anzeigedaten für das ListControl verwaltet, muss lediglich das Interface ListDataModel implementiert. Es erweitert eine bestehende Klasse um Methoden zur Abfrage der Zeilenobjekte innerhalb der Liste. Das Interface ist damit einfach gehalten.

JAVA

```java
import com.cc.framework.ui.model.ListDataModel;

/**
 * Collection with UserDsp-Objects
 */
public class UserDisplayList implements ListDataModel {

    private UserDsp[] data = new UserDsp[0];

    public UserDisplayList(UserDsp[] elements) {
        this.data = elements;
    }

    public Object getElementAt(int index) {
        return data[index];
    }

    public int size() {
        return data.length;
    }

    /**
     * Unique Key for each Row (Object).
     * In this Example our Key only contains the UserId.
     */
    public String getUniqueKey(int index) {
        return data[index].getUserId();
    }
}
```

JAVA

```java
import com.cc.framework.common.DisplayObject;
import com.cc.sampleapp.common.UserRole;

/**
 * User DisplayObject (ViewHelper)
 */
public class UserDsp implements DisplayObject {

    private String userId = "";
    private String firstName = "";
    private String lastName = "";
    private UserRole role = UserRole.NONE;

    public UserDsp(String userId, String firstName,
        String lastName, UserRole role) {

        super();
        this.userId = userId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.role = role;
```

```
    }

    public UserRole getRole()   { return role; }
    public String getUserId()   { return userId; }
    public String getLastName() { return lastName; }
    public String getName() { return firstName + ", " + lastName; }
}
```

1.6 Konfiguration des ListControls innerhalb der JSP-Seite

Um das ListControl-Tag auf einer JSP Seite einzusetzen, muss am Anfang der Seite die entsprechende Tag Library deklariert werden. Anschließend können die Common-Controls mit dem Präfix <ctrl:tagname /> verwendet werden. [Zudem muss die Aufnahme der Tag Bibliotheken im Deployment-Deskriptor, der WEB-INF/web.xml Datei, erfolgen]

JSP

```jsp
<%@ taglib uri="/WEB-INF/tlds/cc-controls.tld" prefix="ctrl" %>

<ctrl:list
    id="userlist1"
    action="sample101/userBrowse"
    name="users"
    title="User List"
    width="500"
    rows="15"
    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>
```

Damit sind alle notwendigen Schritte zur Nutzung des ListControls abgeschlossen.

- Der Blättermechanismus muss nicht selbst implementiert werden, da dieser bereits mit dem ListControl zur Verfügung gestellt wird.
- In der JSP-Seite haben wir festgelegt, dass das ListControl maximal 15 Zeilen zeichnen soll. Wenn mehr Zeilen vorhanden sind, werden automatisch die Buttons zum Vor- und Rückwärtsblättern eingeblendet.
- Bei einem Klick auf den Vorwärtsbutton erfolgt ein Serverroundtrip und die Anzeige der nächste Seite. Die Aktualisierung wird dabei von dem Präsentationsframework übernommen. Es muss kein zusätzlicher Code implementiert werden.

1.7 Tour Ende

Das ListControl lässt sich schnell und einfach integrieren. Dabei ist es noch flexibel genug, um spezielle Anforderungen zu erfüllen. Hierzu kann das Standardverhalten des Kontrollelements überschrieben werden. So lässt sich auch ein dynamisches Laden der Daten realisieren oder es können eigene ListControl-Klassen erstellt werden, die bereits den Zugriff auf eine bestimmte Tabelle kapseln.

Über die Konfiguration in der JSP-Seite lassen sich schnell neue Spalten hinzufügen, die sich zusätzlich berechtigungsabhängig steuern lassen. Der HTML-Code wird über einen Painter erzeugt. Andere Layouts lassen sich durch eine Anpassung der Painter realisieren. Dabei können verschiedene Layouts auch parallel verwendet werden.

Features des ListControls:

- Implementiert einen Blättermechanismus. Kein zusätzlicher Programmieraufwand nötig! Die Buttons am Anfang oder Ende werden automatisch deaktiviert bzw. aktiviert.
- Spaltentypen: Drilldown, Text, CheckBox, Image, Link, Button, Select, Add, Edit, Delete, Control.
- Die Checkspalte unterstützt die Modi "single" und "multiple".
- Buttons konfigurierbar und getrennt ein- und ausblendbar.
- Design des ListControls in der JSP oder auch serverseitig definierbar.
- Bildet Aktion, die auf der Tabelle ausgeführt werden auf CallBack-Methoden in der Action-Klasse ab (Beispiele: onDrilldown, onSort, onEdit, onDelete, onRefresh, onCheck).
- JavaScript Eventhandler auf Spalten hinterlegbar.
- Berechtigungsabhängige Steuerung des Spaltenaufbaus.
- Standardverhalten überschreibbar.
- Layout durch Painterfactory an eigenen StyleGuide (Corporate Identity) anpassbar.
- Optimierter HTML-Code.
- Gleiches Look and Feel in Microsoft® InternetExplorer > 5.x und Netscape™ Navigator > 7.x

1.8 Exkurs: Implementierung einer Callback-Methode

Die Tabelle verwendet bei der Anzeige der UserId eine spezielle Spalte; die Drilldownspalte. Diese löst einen Hyperlink aus, der in unserer Anwendung zu einer Verzweigung in die Detailansicht führt. In dieser Sicht sollen die Daten nicht bearbeitet werden. Hierzu dient der Edit-Button.

Um auf dieses Ereignis entsprechend zu reagieren, nehmen wir nun eine entsprechende CallBack-Methode in unserer UserBrowseAction auf. Da wir die BusinessLogik nicht an dieser Stelle implementieren möchten, leiten wir das Ereignis an eine andere Action - UserDisplayAction - weiter. Diese kann dann die Detailinformationen laden und die entsprechende JSP-Seite zur Anzeige aufrufen.

JAVA

```java
import java.io.IOException;
import javax.servlet.ServletException;

import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.ui.control.ControlActionContext;
import com.cc.framework.ui.control.SimpleListControl;

import com.cc.sampleapp.common.Forwards;
import com.cc.sampleapp.common.Messages;
import com.cc.sampleapp.dbaccess.DBUser;
import com.cc.sampleapp.presentation.dsp.UserDisplayList;

public class UserBrowseAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx)
        throws IOException, ServletException {

    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();
    }

    /**
     * This Method is called when the Drilldown-Column is clicked
     * In our Example we switch to the DetailView, which shows
     * more Information about the User. It's a readonly View.
     * @param   ctx ControlActionContext
     * @param   key     UniqueKey, as it was defined in the UserDisplayList
     *          to identify the Row. In this Example the UserId.
     */
    public void users_onDrilldown(ControlActionContext ctx, String key) {
        ctx.forwardByName(Forwards.DRILLDOWN, key);
    }
}
```

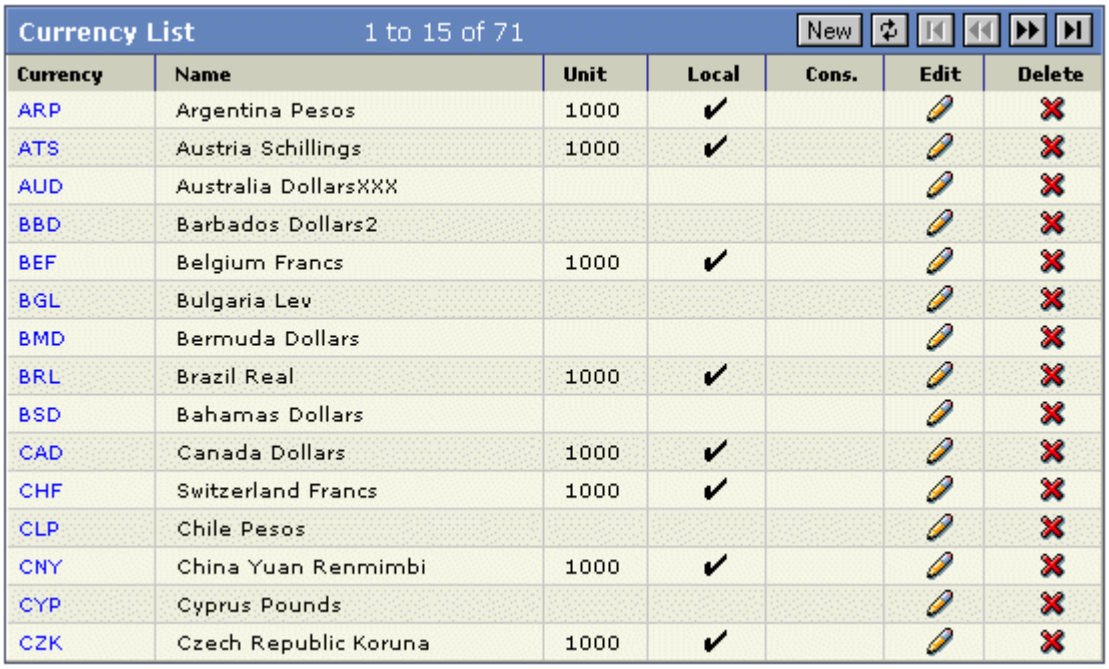
Der Name der CallBack-Methode setzt sich immer aus dem **Namen der Bean**, dem Präfix **_on** und dem eingetretenen **Event** zusammen. Der Name der Bean bestimmt sich wie folgt:

- Wird das ListControl direkt übergeben, z.B. innerhalb der Session (wie oben) --> Dann entspricht der Beanname dem Name des Attributes, unter dem die Bean 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.

In unserem Beispiel wird die Instanz des Kontrollelementes in der Session gehalten, damit das Kontrollelement seinen internen Status (aktuelle Seite, Displaydaten) über mehrere Serverroundtrips hinweg behält. Die zu implementierende CallBack-Methode muss daher den Namen users_onDrilldown tragen. Sie bekommt den ControlActionContext übergeben, der unter anderem den Zugriff auf das Session-, Request- und Response-Objekt kapselt. Als weiterer Parameter wird der eindeutige Schlüssel für die Zeile übergeben, wie er innerhalb der DisplayListe mittels der Methode getUniqueKey(int index) spezifiziert wurde. Unsere UserDisplayList liefert hier die UserId zurück.

1.9 Alternative Layouts

Andere Layouts lassen sich über die Implementierung und Registrierung eigener Painterfactorys erzeugen. Anbei einige Beispiele aus Anwendungsprojekten.



Currency List 1 to 15 of 71 New

Currency	Name	Unit	Local	Cons.	Edit	Delete
ARP	Argentina Pesos	1000	✔			
ATS	Austria Schillings	1000	✔			
AUD	Australia DollarsXXX					
BBD	Barbados Dollars2					
BEF	Belgium Francs	1000	✔			
BGL	Bulgaria Lev					
BMD	Bermuda Dollars					
BRL	Brazil Real	1000	✔			
BSD	Bahamas Dollars					
CAD	Canada Dollars	1000	✔			
CHF	Switzerland Francs	1000	✔			
CLP	Chile Pesos					
CNY	China Yuan Renmimbi	1000	✔			
CYP	Cyprus Pounds					
CZK	Czech Republic Koruna	1000	✔			

Abbildung 2: Layoutbeispiel aus einem Anwendungsprojekt

2 Glossar

CC	Common Controls
JSP	JavaServer Pages
Painter	Erzeugt den HTML-Code eines Kontrollelements; über eine eigene PainterFactory lässt sich das Layout an ein Corporate Design anpassen.
DataModel	Schnittstelle, über die ein Kontrollelement seine Anzeigedaten bezieht.