

Common-Controls Guided Tour ListControl

Version 1.6 - Stand: 14. Januar 2006

Herausgeber:

SCC Informationssysteme GmbH
64367 Mühlthal

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

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

Copyright © 2000 - 2006 SCC Informationssysteme GmbH.
All rights reserved. Published 2003

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way without the prior agreement and written permission of SCC Informationssysteme GmbH.

Sun, Sun Microsystems, the Sun Logo, Java, JavaServer Pages are registered trademarks of Sun Microsystems Inc in the U.S.A. and other Countries.

Microsoft, Microsoft Windows or other Microsoft Produkte are a registered trademark of Microsoft Corporation in the U.S.A. and other Countries.

Netscape, Netscape Navigator is a registered trademark of Netscape Communications Corp in the U.S.A. and other Countries.

All other product names, marks, logos, and symbols may be trademarks or registered trademarks of their respective owners.

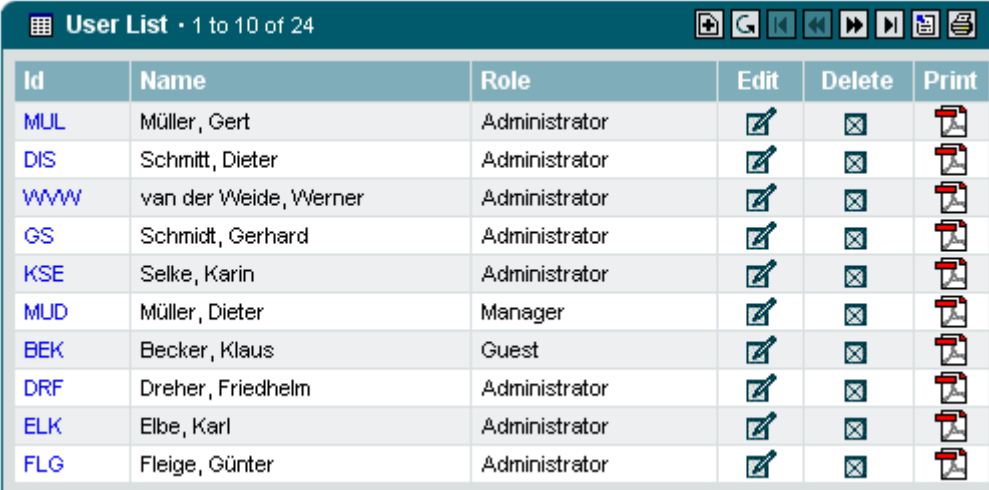
Inhaltsverzeichnis

1	Guided Tour ListControl	1
1.1	Gegenstand.....	1
1.2	Registrierung der Painterfactory	2
1.3	Ableitung der Action Klasse für den Struts-Adapter.....	2
1.4	Instanziierung des ListControls	3
1.5	Bereitstellen der Anzeigedaten	4
1.6	Konfiguration des ListControls innerhalb der JSP-Seite	5
1.7	Tour Ende.....	6
1.8	Exkurs: Implementierung einer Callback-Methode	7
1.9	Anhang: Alternative Layouts	8
2	Glossar	9

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 Datenmodells beim Klick auf die Checkspalte müssen nicht implementiert werden. Diese Grundfunktionen deckt das ListControl bereits ab.



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			

Abbildung 1: Beispieldarstellung ListControl

Zum Einsatz des ListControls sind folgende Schritte notwendig:

1. Auswahl des Designs für die Benutzeroberfläche
2. Erstellung einer Action-Klasse
3. Instanziierung des 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 anwendungsweit in der `init()`-Methode des Frontcontroller-Servlets geschehen.¹ Wir wählen hier das Standarddesign, welches uns der `DefaultPainter` bereitstellt².

```
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());
    }
}
```

1.3 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 „BrowseUserAction“ 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. [\[FWAction ist von org.apache.struts.action.Action abgeleitet\]](#) Sie erhält beim Aufruf den `ActionContext`, über den der Zugriff auf weitere Objekte, wie das Request- und Response-Objekt gekapselt ist.

```
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 Wenn der einzelne Benutzer zwischen verschiedenen Oberflächendesigns wählen können soll, dann werden zusätzliche `PainterFactories` 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.

1.4 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.

```
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 Exception {

        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 „ListModel“ implementiert. Es erweitert eine bestehende Klasse um Methoden zur Abfrage der Zeilenobjekte innerhalb der Liste. Das Interface ist damit einfach gehalten.

```
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();
    }
}
```

```
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. [\[Zudem muss die Aufnahme der Tag Bibliothek im Deployment-Deskriptor, der WEB-INF/web.xml Datei, erfolgen\]](#)

```
<%@ taglib uri="/WEB-INF/tlds/cc-controls.tld" prefix="ctrl" %>
<ctrl:list
  action="sample101/userBrowse"
  name="users"
  title="User List"
  width="500"
  rows="15"
  refreshButton="true"
  createButton="true">

  <ctrl:columnndrilldown
    title="Id"
    property="userId"
    width="65"/>

  <ctrl:columnntext
    title="Name"
    property="name"
    width="350"/>

  <ctrl:columnntext
    title="Role"
    property="role.value"
    width="150"/>

  <ctrl:columnndedit
    title="Edit"/>

  <ctrl:columnnddelete
    title="Delete"/>

</ctrl:list>
```

Damit sind alle notwendigen Schritte zur Nutzung des ListControls abgeschlossen. Der Blättermechanismus muss nicht selbst implementiert werden, da dieser bereits von 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 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.

```
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 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();
    }

    /**
     * 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)
        throws Exception {
        ctx.forwardByName(Forwards.DRILLDOWN, key);
    }
}
```

Der Name der Callback-Methode setzt sich dabei aus dem Property-Namen des ListControls – dem Namen der Bean - und dem eingetretenen Event zusammen. Da das ListControl unter dem Namen „users“ in der Session abgelegt wurde, lautet der Name der Callback-Methode **users_onDrilldown**.

1.9 Anhang: 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 1

2 Glossar

C

CC

Common-Controls