Common-Controls Guided Tour ListControl

Version 1.0.3 – Last changed: 01. August 2004

Publisher:
SCC Informationssysteme GmbH
64367 Mühltal (Germany)

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

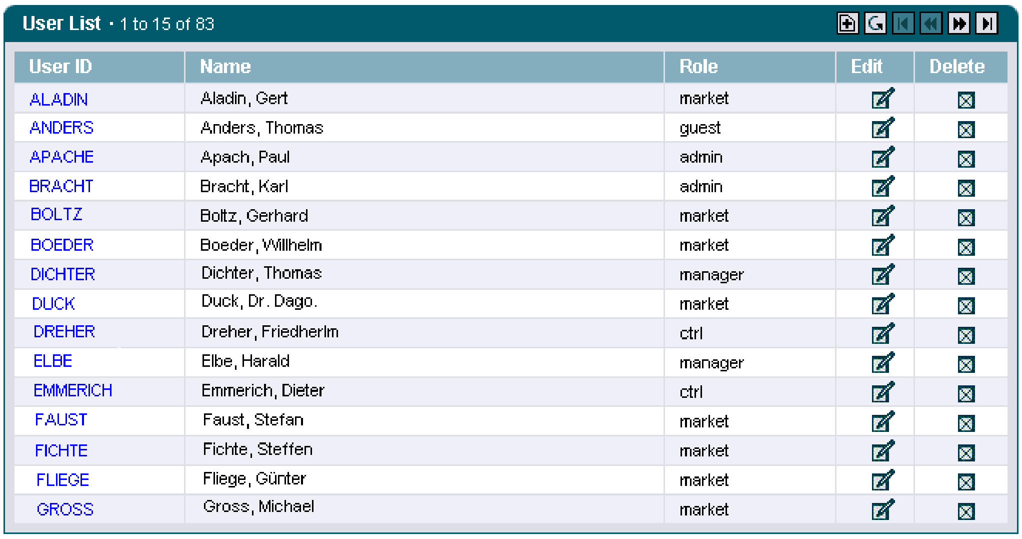
Product Site:
www.common-controls.com

Table of contents

1 Guided Tour ListControl

1.1 Object

This exercise demonstrates the use of the ListControl. The control element generates a table whose structure and appearance can be freely configured. The scrolling mechanism, the sorting within the columns or the updating of the data model on clicking the check column do not have to be implemented. These basic functions are already covered by the ListControl.


User List · 1 to 15 of 83

User ID	Name	Role	Edit	Delete
ALADIN	Aladin, Gert	market	✎	☒
ANDERS	Anders, Thomas	guest	✎	☒
APACHE	Apach, Paul	admin	✎	☒
BRACHT	Bracht, Karl	admin	✎	☒
BOLTZ	Boltz, Gerhard	market	✎	☒
BOEDER	Boeder, Willhelm	market	✎	☒
DICHTER	Dichter, Thomas	manager	✎	☒
DUCK	Duck, Dr. Dago.	market	✎	☒
DREHER	Dreher, Friedherlm	ctrl	✎	☒
ELBE	Elbe, Harald	manager	✎	☒
EMMERICH	Emmerich, Dieter	ctrl	✎	☒
FAUST	Faust, Stefan	market	✎	☒
FICHTE	Fichte, Steffen	market	✎	☒
FLIEGE	Fliege, Günter	market	✎	☒
GROSS	Gross, Michael	market	✎	☒

Figure 1: Example ListControl

The following steps are required for using the ListControl:

1. Selection of the design for the user interface
2. Generation of an action class
3. Instancing a ListControl
4. Provision of display data
5. Configuration of the TreeControls within the JSP-Page

1.2 Registration of the Painterfactory

The registration of the Painterfactory takes place first. It defines which design the user interface will get. This can be done across the application in the init()-method of the Frontcontroler-Servlet.[1] Here, we select the standard design that the DefaultPainter offers us.[2]

```
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 Derivation of the Action class for the Struts adapter

Our table has to display information on the system users. Therefore, the action class which takes care of the loading and filling of the ListControl must have the nomenclature "UserBrowseAction". The action class is then derived from the class FWAction, which the Struts-action class encapsulates and extends with functionalities of the presentation framework. Instead of the execute()-method, the doExecute()-method is called. On calling, it gets the ActionContext, through which the access to additional objects such as the Request- Session- and Response-object is capsulated.

```
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] If it has to be possible for the individual user to choose between different interface designs, then additional PainterFactorys are registered in the user session. This is done mostly in the LoginAction with PainterFactory.registerSessionPainter() in the session Scope.

[2] Additional designs (PainterFactories) are included in the kit of the Professional Edition, or you can develop them yourself.

1.4 Instancing the ListControl

Now, the ListControl is instanced within the UserBrowseAction and filled with the data to be displayed. To do so, using the setDataModel() method, its data model is assigned to the control element. The method setDataModel() takes up, as an argument, a **ListDataModel**. What is involved here is a simple interface, which is implemented by the class UserDisplayList, which provides the display data.

```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 Provision of the display data

The class that administers the display data for the ListControl must only implement the interface **ListDataModel**. It extends an existing class with methods for querying the line objects within the list. The interface is thus kept simple.

```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
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 Configuration of the ListControl within the JSP-Page

In order to use the ListControl tag on a JSP page, the corresponding tag library must be declared at the start of the page. Then, the Common-Controls can be used with the prefix <ctrl:*tagname*/>. [In addition, the tag libraries must be included in the Deployment descriptor, the WEB-INF/web.xml file]

```
<%@ 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>
```

All the necessary steps for using the ListControl are thus complete.

- The scrolling mechanism does not have to be self-implemented, since it is already made available by the ListControl.
- In the JSP-Page, we have determined that the ListControl should draw a maximum of 15 lines or rows. When more lines are available, the buttons for scrolling forward and back are displayed automatically. A click on the forward button results in a Server roundtrip and the display of the next page. The refreshing is then taken care of by the presentation framework. No additional code has to be implemented.

1.7 Tour End

The ListControl can be easily and quickly integrated. But it is still flexible enough to fulfill special requirements. For this, the standard behavior of the control element can be overwritten. Thus, it is also possible to dynamically load the data, or separate ListControl classes can be generated, which already capsulate the access to a certain table.

Using the configuration in the JSP-Page, new columns can be quickly added, which can be additionally controlled in an authorization-dependent manner. The HTML code is generated with a Painter. However, other layouts can also be implemented by customizing the Painter. Various layouts can be used in parallel.

Features of the ListControls:

- Implements a scrolling mechanism. No additional programming effort required. The buttons at the start or the end are automatically disabled or enabled.
- Column types: Drilldown, Text, CheckBox, Image, Link, Button, Select, Add, Edit, Delete, Control.
- The Checkcolumn supports the modes "single" and "multiple".
- Buttons are configurable and can be hidden or displayed individually.
- Design of the ListControl can be defined in the JSP page or also on the server side.
- Maps the action that is to be carried out on the table to CallBack methods in the action class (Examples: onDrilldown, onSort, onEdit, onDelete, onRefresh, onCheck).
- JavaScript Eventhandler can be saved in columns.
- Authorization-dependent control of the column construction.
- Standard behavior can be overwritten.
- Layout through Painterfactory can be matched to own StyleGuide (Corporate Identity).
- Optimized HTML-Code.
- Same Look and Feel in Microsoft® InternetExplorer > 5.x and Netscape™ Navigator > 7.x

1.8 Exkurs: Implementation of a Callback method

The table uses a special column for displaying the detail data; the drilldown column. A click on this column triggers a hyperlink which, in our application, results in a branch to the detail view. The data should not be processed in this view. The Edit button is used for this.

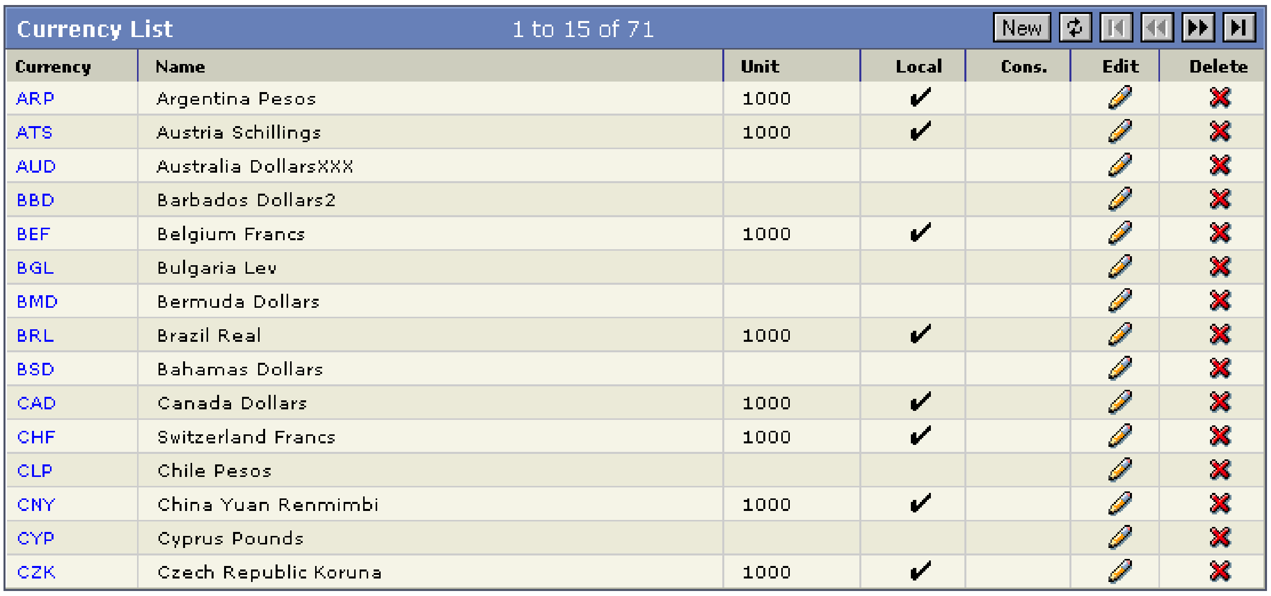
To react suitably to this event, we include a corresponding Callback method in our UserBrowseAction. Since we do not wish to implement the business logic here, we forward the event to another action - UserDisplayAction. This can then load the detailed information and call the corresponding JSP-Page for display.

```
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);
       }
}
```

The name of the CallBack method is composed of name of the ListControl - the name of the Bean - and the event that has occurred. Since the ListControl was saved the Session under the name "users" the name of the CallBack method is **users**_onDrilldown.

1.9 *Alternative Layouts*

Other layouts can be generated by means of the implementation and registration of own Painterfactorys. Here are a few examples from application projects.



Currency List 1 to 15 of 71

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	✓		✎	✗

Figure 2: Example Layout 1

2 Glossary

C

CC	Common-Controls