Common-Controls Guided Tour TreeListControl

Version 1.6 - 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 Managed Tour TreeListControl

1.1 Gegenstand

Diese Übung demonstriert den Einsatz des TreeListControls. Es kombiniert einen Baum mit einer Liste, dessen Knoten sich auf- und zugeklappen lassen. Der Programmierer stellt hierfür lediglich die Anzeigedaten (das Datenmodell) durch Implementierung eines einfachen Interfaces bereit.

Das TreeListControl bietet die folgenden Features:

- Die Linien auf der obersten Ebene können ein oder ausgeblendet werden. Zu den Knoten und den Blättern lassen sich unterschiedliche Bilder in einer Image Map hinterlegen. Dabei wird die Zuordnung der Bilder zu dem jeweiligen Baumknoten mit Hilfe von Regulären Ausdrücken vorgenommen.
- Das Kontrollelement verwaltet selbstständig alle notwendigen Zustandsdaten über mehrere Server Roundtrips hinweg. Dazu zählt beispielsweise der auf- oder zugeklappte Status der Baumknoten.
- Vor den Baumeinträgen können Checkboxen ein oder ausgeblendet werden. Bei der Auswahl eines Knotens auf einer unteren Ebene, werden automatisch alle übergeordneten Knoten markiert.

Zum Einsatz des TreeListControls sind folgende Schritte notwendig:

- Auswahl des Layouts der Benutzeroberfläche
- Ableitung einer Actionklasse
- Instanziierung eines TreeListControls
- Bereitstellung der Anzeigedaten
- Konfiguration des Kontrollelementes 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 Frontcontroler-Servlets geschehen.[1] Wir wählen hier das Standarddesign, welches uns der DefaultPainter bereitstellt[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 Ableitung der Action Klasse

In unserem TreeListControl wollen wir Regionen und Länder anzeigen. Daher soll die Action-Klasse, welche das Laden und Befüllen des TreeListControls übernimmt, die Bezeichnung „RegionBrowseAction“ tragen. Die Actionklasse wird 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 abgeleitet] Sie erhält beim Aufruf den ActionContext, über den der Zugriff auf weitere Objekte, wie das Request- und Response-Objekt gekapselt ist.

```
import java.lang.Exception;

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

public class RegionBrowseAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx) throws Exception {
        // In the next chapter, we will instantiate
        // our TreeListControls 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 TreeListControls

Nun wird das TreeListControl innerhalb unserer Action instanziiert und mit den Anzeigedaten gefüllt. Das Datenmodell wird dem Kontrollelement über die setDataModel()-Methode zugeordnet. Die Methode nimmt als Argument ein Objekt vom Typ **TreeGroupDataModel** entgegen. Dabei handelt es sich um ein Interface welches den Zugriff auf die Anzeigedaten des Baumes bereitstellt. Es ist die Aufgabe des Anwendungsentwicklers eine entsprechende Implementierung zur Verfügung zu stellen.

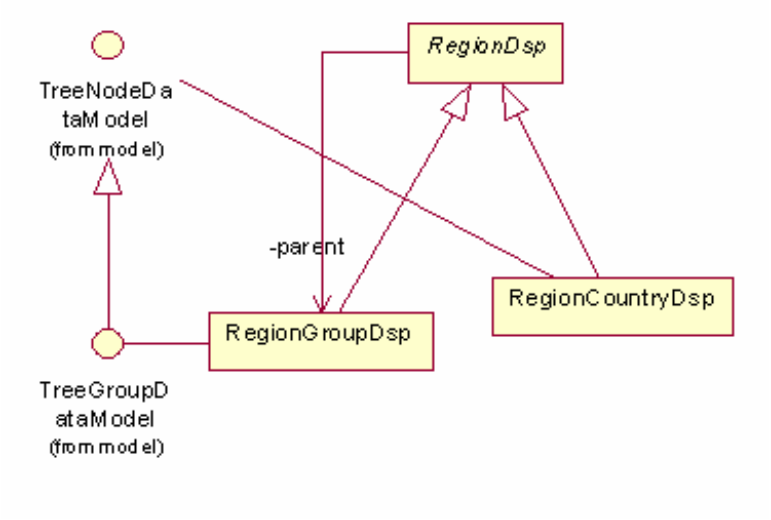
```
import java.lang.Exception;

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

public class RegionBrowseAction extends FWAction {

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

        try {
            // first get the Displaydata for our TreeList
            RegionGroupDsp dspData = DBRegion.fetchDspOutline();

            // Create the TreeListControl an populate it
            // withe the Data to display
            TreelistControl regionList = new TreelistControl();
            regionList.setDataModel(dspData);

            // third put the TreeListControl into the Session-Object.
            // Our TreeListControl is a statefull Object.
            // Normaly  you can use an Objectmanager or an other
            // workflow Component that manage the Livecyle of the Object
            ctx.session().setAttribute("regions", regionList);
        }

        catch (Throwable t) {
            ctx.addGlobalError("Error: ", t);
        }

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

1.5 Bereitstellen der Anzeigedaten

Der Baum besteht aus Gruppen- und Blattknoten. Gruppenknoten können wieder weitere Knoten enthalten (Composite Pattern). Entsprechend stehen für die beiden Knotentypen die Interfaces **TreeGroupDataModel** und **TreeNodeDataModel** bereit (TreeGroupDataModel erweitert dabei TreeNodeDatamodel). Mit ihrer Hilfe lässt sich die Baumstruktur einfach erzeugen.

Dabei wird zuerst der Wurzelknoten erzeugt unter dem dann weitere Gruppen oder Blätter eingehängt werden. Der Wurzelknoten wir dem TreeListControl als Datenmodell übergeben.

Die Vorgehensweise entspricht der Bereitstellung der Anzeigedaten für das TreeControl. Im Unterschied zu dem TreeControl soll das TreeListControl jedoch noch weitere Spalten präsentieren. Dazu muss unsere Bean, welche die Anzeigedaten bereitstellt, lediglich weitere Properties für die entsprechenden Spalten implementieren. In unserem Beispiel handelt es sich dabei um die Klasse RegionDsp, von der Gruppen- und Blattknoten abgeleitet sind.



Ein ausführliches Code-Beispiel wird Ihnen mit der Trialversion bereitgestellt, die Sie kostenlos downloaden können.

1.6 Konfiguration des Layouts innerhalb der JSP-Seite

Um das TreeListControl-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* /> referenziert 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:treelist
    name="regions"
    action="sample301/regionBrowse"
    title="Regions Structure"
    rows="15"
    refreshButton="true"
    expandMode="multiple"
    root="true">

    <ctrl:columntree
        title="Region"
        property="region"
        width="180"
        imageProperty="type"/>

    <ctrl:columntext
        title="Name"
        property="name"
        width="250"/>

    <ctrl:columnadd
        title="Add"
        property="add"/>

    <ctrl:columnedit
        title="Edit"
        property="editable"/>

    <ctrl:columndelete
        title="Delete"
        property="editable"/>
</ctrl:treelist>
```

Da wir das TreeListControl in der Session abgelegt haben, wird der Name der Bean über das name-Attribut angegeben. Zudem wird über das action-Attribut die Action spezifiziert, an die Aktionen aus unserem TreeListControl (onEdit, onDelete, etc...) delegiert werden.

Wird das TreeListControl in einer FormBean abgelegt, reicht die Angabe des property-Attributes aus. Der Scope der Formbean muss in diesem Fall auf „session“ eingestellt werden, damit das Kontrollelement über Server Roundtrips hinweg seinen Zustand behalten kann.

Bei Einsatz einer Workflowsteuerung kann das Kontrollelement über andere Komponenten erzeugt und später wieder aus der Session gelöscht werden.

Damit sind alle notwendigen Schritte zur Nutzung des TreeListControls abgeschlossen.
Das Auf- und Zuklappverhalten muß nicht selbst implementiert werden. Es wird durch das Kontrollelement selbst verwaltet. Zur Navigation stellt das Kontrollelement Blätterbuttons zur Verfügung, die aktiviert werden, sobald die vorgegebene Zeilenanzahl überschritten wird.

1.7 Tour Ende

Das TreeListControl lässt sich schnell und einfach integrieren. Sein Standardverhalten lässt sich bei Bedarf auch überschreiben. So lassen sich auch spezielle TreeListList-Objekte erstellen, die bereits den Zugriff auf bestimmte fachliche Daten kapseln und wiederkehrend innerhalb eines Anwendungsprojektes verwendet werden können.

Durch die Konfigurationsmöglichkeiten in der JSP-Seite kann das Verhalten des TreeListControls schnell geändert werden. Alternative Designs lassen sich durch eine Anpassung der bestehenden Painter einfach integrieren. Dabei werden unterschiedliche Designs auch parallel unterstützt.

Der Programmierer kann sich auf die fachlichen Abläufe und die Bereitsstellung der Anzeigedaten konzentrieren.

Features des TreeListControls:

- Implementiert ein automatisches auf- und zuklappen von Knoten
- Verwaltet den Zustand von optionalen Checkboxen.
- Verschiedene Konfigurationsmöglichkeiten (Ein- oder Ausblenden des Wurzelknotens, Änderung des Auf- und Zuklappverhalten einstellbar, Verbindungslinien auf oberster Ebene ausblendbar)
- Daten unterhalb eines Gruppenknotens können auch erst bei Öffnen der Gruppe geladen werden. Der Baum muss also nicht von Anfang an vollständig bekannt sein. Dies ist beispielsweise in Verbindung mit Datenbanken hilfreich. Beim ersten Aufklappen eines Knoten mit unbekannter Kinderzahl wird der Anwendung ein onExpandEx Ereignis gesendet.
- Design des TreeListControls in der JSP oder auch serverseitig definierbar!
- Bildet Aktion, die auf dem Baum ausgeführt werden auf CallBack-Methoden in der Action-Klasse ab (Beispiele: onCheck, onExpand, onCollapse, onExpandEx).
- Bilder vor den Knoten/Blättern über Reguläre Ausdrücke zuordbar.
- Berechtigungsprüfung auf Knotenebene. Knoten können damit automatisch für unberechtigte Anwender ausgeblendet werden. (siehe Scurity Dokumentation)
- Design 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 von Callback-Methoden

Das TreeListControl generiert bei einem Klick auf ein **Label** automatisch ein onDrilldown-Event auf das der Programmierer innerhalb der Action-Klasse reagieren kann.

Um in unserem Beispiel auf dieses Ereignis zu reagieren, nehmen wir eine entsprechende Callback-Methode in der RegionBrowseAction auf. Da wir die Businesslogik nicht an dieser Stelle implementieren möchten, leiten wir das Ereignis an eine andere Action – RegionDisplayAction – weiter.

```
import java.lang.Exception;

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

public class RegionBrowseAction extends FWAction {

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

        try {
              RegionGroupDsp dspData = DBRegion.fetchDspOutline();

              TreelistControl regionList = new TreelistControl();
              regionList.setDataModel(dspData);

              ctx.session().setAttribute("regions", regionList);
        }

        catch (Throwable t) {
              ctx.addGlobalError("Error: ", t);
        }

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

      // ------------------------------------------------
      //          TreeList-Control Event Handler
      // ------------------------------------------------

      /**
       * This Method is called when the TreeLabel is clicked
       * In our Example we switch to the DetailView, which shows
       * more Information about the node.
       * @param    ctx    ControlActionContext
       * @param    key    UniqueKey, as created in the Datamodel
       */
      public void regions_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 TreeListControls – dem Namen der Bean - und dem eingetretenen Event zusammen. Da das TreeListControl unter dem Namen „region“ in der Session abgelegt wurde, lautet der Name der CallBack-Methode **regions**_onDrilldown.

1.9 Anhang: Alternative Layouts

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

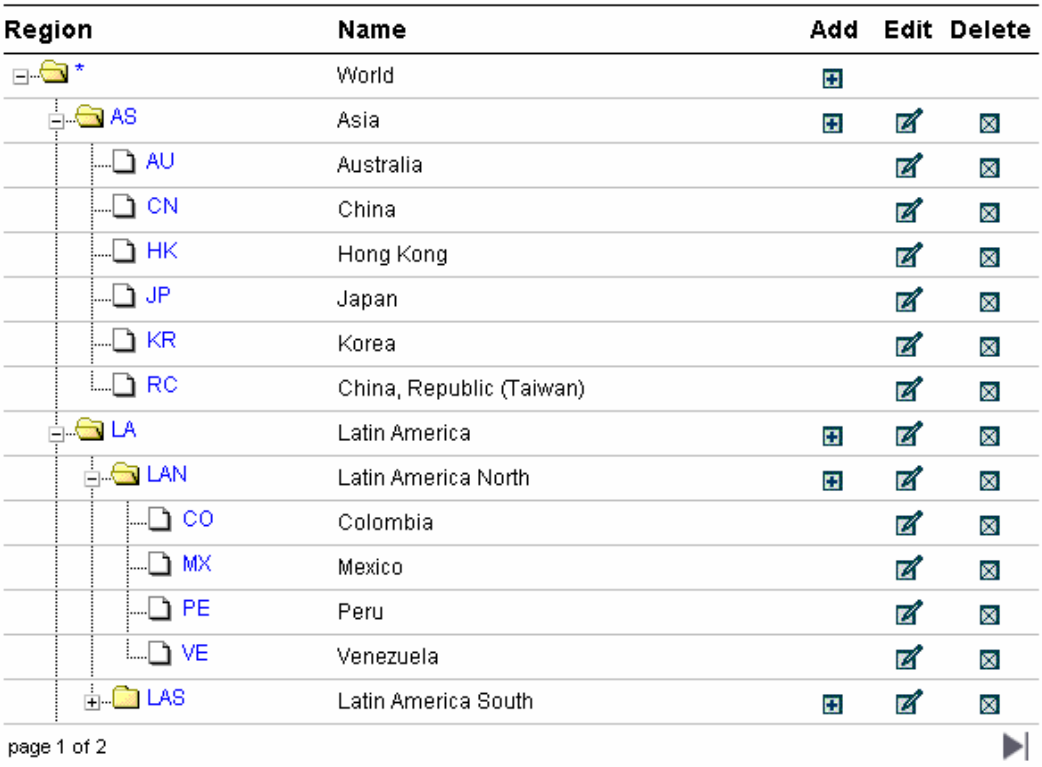


Abbildung 1: Layoutbeispiel 1

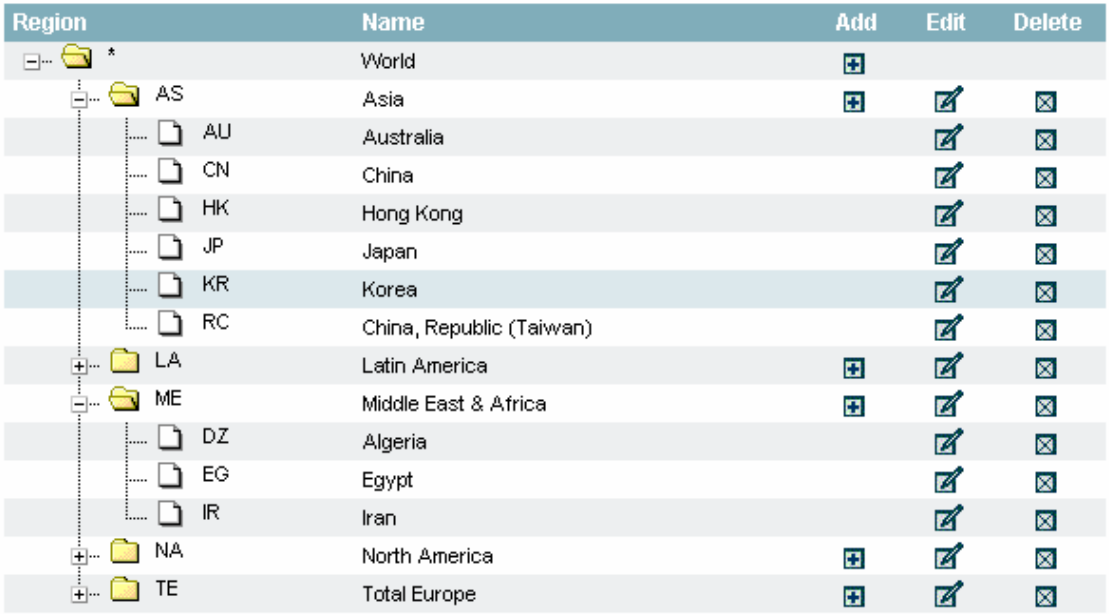


Abbildung 2: Layoutbeispiel 2

2 Glossar

C

CC

Common-Controls