SCC
INFORMATIONSSYSTEME

COMMON CONTROLS · GUIDED TOUR

TreeControl

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 TreeControl

1.1 Gegenstand

Diese Übung demonstriert den Einsatz des TreeControls. Dieses Kontrollelement erzeugt einen Baum, dessen Knoten sich auf- und zugeklappen lassen. Der Programmierer stellt hierfür lediglich die Anzeigedaten (das Datenmodell) durch Implementierung eines einfachen Interfaces bereit.

Das TreeControl 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 ImageMap 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.

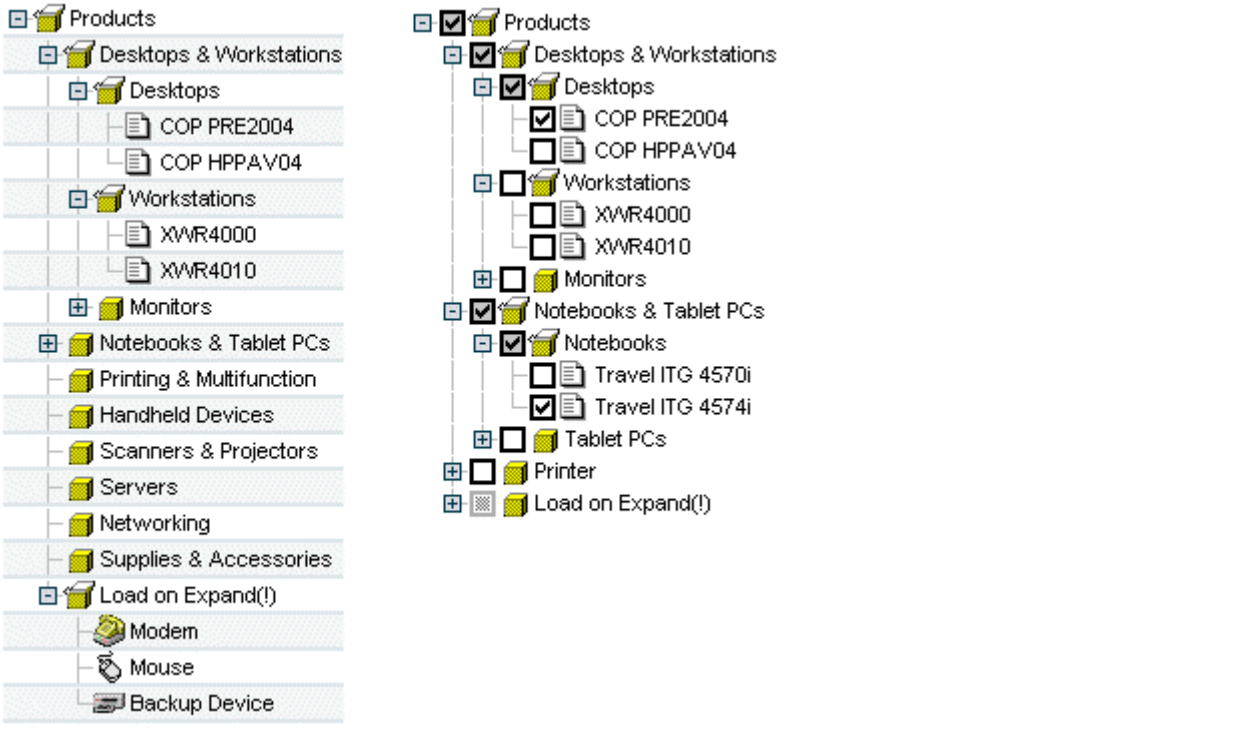

Abbildung 1: Beispieldarstellung TreeControl

Zum Einsatz des TreeControls sind lediglich folgende Schritte notwendig:

1. Auswahl des Designs der Benutzeroberfläche
2. Erstellung einer Actionklasse
3. Instanziierung eines TreeControls
4. Bereitstellung der Anzeigedaten

5. Konfiguration des Baumes 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-Layout
        // In this case we only use the Default-Layout.
        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

In unserem Baum wollen wir Produktgruppen und Produkte anzeigen. Daher soll die Action-Klasse, welche das Laden und Befüllen des TreeControls übernimmt, die Bezeichnung "ProductTreeBrowseAction" 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. 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 ProductTreeBrowseAction 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 TreeControls with the DisplayData
    }
}
```

1.4 Instanziierung des TreeControls

Nun wird das TreeControl 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.

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.TreeControl;

public class ProductTreeBrowseAction extends FWAction {

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

        try {
            // first we get the Data for our Tree
            ProductGroupDsp data = DBProduct.fetch();

            // secondly create the TreeControl and populate it
        // with the Data to display
            TreeControl products = new TreeControl();
            products.setDataModel(data);

        // third put the TreeControl into the Session-Object.
            // Our Control is a statefull Object.
            ctx.session().setAttribute("products", products);
        }

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

        // Display the Page with the Tree
        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 wird dem TreeControl als Datenmodell übergeben.

JAVA

```java
// Root
ProductGroupDsp root = new ProductGroupDsp("0", "Products", "Root");

ProductGroupDsp group = null;
ProductGroupDsp subgroup = null;

// First Group under the Root-Element
group = new ProductGroupDsp("1201", "Workstations & Monitors");

subgroup = new ProductGroupDsp("2102", "Workstations");
subgroup.addChild( new ProductDsp("3005", "XWR4000", "product description"));
subgroup.addChild( new ProductDsp("3005", "XWR4010", "product description"));
group.addChild(subgroup);

subgroup = new ProductGroupDsp("2103", "Monitors");
subgroup.addChild( new ProductDsp("3101", "ITV 2800") );
subgroup.addChild( new ProductDsp("3102", "ITV 2820i") );
subgroup.addChild( new ProductDsp("3103", "ITV 3220") );
group.addChild(subgroup);

root.addChild(group);

// Secound Group under the Root-Element
group = new ProductGroupDsp("1204", "Printing & Multifunction");
root.addChild(group);
```

Die folgenden Klassen bilden die entsprechenden Modelle für dieses Beispiel ab:

- ProductGroupDsp
- ProductDsp
- ProductBaseDsp

1.6 Konfiguration des TreeControls innerhalb der JSP-Seite

Um das TreeControl-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:tree
    id="prodtree1"
    name="products"
    action="sample201/productBrowse"
    root="true"
    linesAtRoot="true"
    labelProperty="name"
    imageProperty="type"
    expandMode="multiple"
    groupselect="true"
    checkboxes="false"/>
```

Damit sind alle notwendigen Schritte zur Nutzung des TreeControls abgeschlossen. Das Auf- und Zuklappverhalten muss nicht selbst implementiert werden, es wird durch das Kontrollelement verwaltet. Dies gilt auch für die Selektionszustände von Checkboxen, die über das Attribut checkboxes="true" aktiviert werden können. Der Programmierer kann sich also auf die fachlichen Abläufe und die Bereitstellung der Anzeigedaten konzentrieren.

Professional Edition: Mit dem Attribut runat="client" wird der Baum in einer JavaScript Version generiert und kann somit ohne Server Roundtrips auf- und zugeklappt werden. Diese Erhöhung des Benutzerkomforts bedarf jedoch keinerlei Änderungen im Anwendungsprogramm.

1.7 Tour Ende

Das TreeControl lässt sich schnell und einfach integrieren. Sein Standardverhalten lässt sich bei Bedarf überschreiben. So lassen sich auch spezielle TreeControl-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 TreeControls schnell geändert werden. Alternative Designs lassen sich durch eine Anpassung der bestehenden Painter einfach integrieren. Dabei werden unterschiedliche Designs auch parallel unterstützt.

Features des TreeControls:

- 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 TreeControls 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 Security-Dokumentation).
- 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

Das TreeControl 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 ProductTreeBrowseAction auf. Da wir die BusinessLogik nicht an dieser Stelle implementieren möchten, leiten wir das Ereignis an eine andere Action - ProductDisplayAction - weiter.

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.TreeControl;

import com.cc.sampleapp.common.Forwards;

public class ProductTreeBrowseAction extends FWAction {

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

        try {
            ProductGroupDsp data = DBProduct.fetch();
            TreeControl products = new TreeControl();
            products.setDataModel(data);
            ctx.session().setAttribute("products", products);
        }

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

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

    // ------------------------------------------------
    //          Tree-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  products_onDrilldown(ControlActionContext ctx, String key) {
        ctx.forwardByName(Forwards.DRILLDOWN, key);
    }

}
```

Der Name der CallBack-Methode setzt sich dabei aus dem Property-Namen des TreeControls - dem Namen der Bean - und dem eingetretenen Event zusammen. Da das TreeControl unter dem Namen "products" in der Session abgelegt wurde, lautet der Name der CallBack-Methode products_onDrilldown.

1.9 Exkurs: Verwendung einer ImageMap

Das TreeControl verwendet bei der Darstellung für geöffnete und geschlossene Knoten vordefinierte Bilder, die sich bei Bedarf einfach austauschen lassen. Jedem Eintrag innerhalb des Baumes kann ein eigenes Bild zugeordnet werden. Eine Möglichkeit hierzu stellt die Verwendung einer ImageMap dar. Diese wird außerhalb des Baumes in der JSP-Seite deklariert und dem TreeControl über das Attribut "imagemap" zugeordnet.

JSP

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

<util:imagemap name="imap_products">
    <util:imagemapping
        rule="prodgroup.open"
        src="images/imgBoxOpen.gif"
        width="16" height="16"/>
    <util:imagemapping
        rule="prodgroup.closed"
        src="images/imgBoxClosed.gif"
        width="16" height="16"/>
    <util:imagemapping
        rule="product"
        src=" images/imgItem.gif"
        width="16" height="16"/>
</util:imagemap>

<ctrl:tree
    id="prodtree1"
    name="products"
    action="sample201/productBrowse"
    root="true"
    linesAtRoot="true"
    labelProperty="name"
    imageProperty="type"
    imagemap="imap_products"
    expandMode="multiple"
    groupselect="true"
    checkboxes="true"/>
```

Arbeitsweise: Bei der Darstellung des Baumes liefert das Datenmodel über die Methode getType() jeweils einen Ausdruck zurück, der mit der ImageMap verglichen wird. Bei einer Übereinstimmung mit einer Regel wird das entsprechende Bild gezeichnet. Für geschlossene und geöffnete Knoten, werden diese Ausdrücke automatisch um das Suffix ".open" bzw. ".closed" erweitert, so dass sich unterschiedliche Bilder für die unterschiedlichen Zustände verwenden lassen. Die Angabe der Regeln erfolgt mit regulären Ausdrücken.

Die Methode, welche den Ausdruck für das zuzeichnende Bild liefert, wird über das Attribut "imageProperty" bestimmt. In unserem Beispiel wird das type-Property verwendet, welches für Gruppen stets "prodgroup" und für einzelne Blätter stets "product" liefert.

1.10 Alternative Layouts

Andere Layouts lassen sich über die Implementierung und Registrierung eigener Painterfactorys erzeugen, wie das folgende Beispiel zeigt:

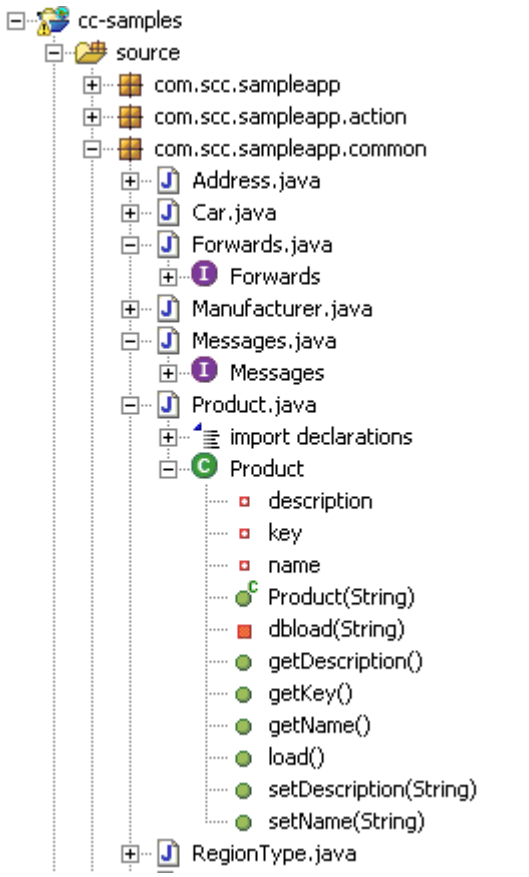

Abbildung 2: Layoutbeispiele mit eigener PainterFactory

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.